Verification of the Randomized Consensus Algorithm of Aspnes and Herlihy: a Case Study*

Anna Pogosyants[1] Roberto Segala[2] Nancy Lynch[1]

[1] Laboratory for Computer Science, MIT
[2] Dipartimento di Scienze dell'Informazione, Università di Bologna

Abstract. The Probabilistic I/O Automaton model of [11] is used as the basis for a formal presentation and proof of the randomized consensus algorithm of Aspnes and Herlihy. The algorithm is highly nontrivial and guarantees termination within expected polynomial time. The task of carrying out this proof has led us to develop several general proof techniques for probabilistic I/O automata. These include ways to combine expectations for different complexity measures, to compose expected complexity properties, to convert probabilistic claims to deterministic claims, to use abstraction mappings to prove probabilistic properties, and to apply random walk theory in a distributed computational setting.

1 Introduction

With the increasing complexity of distributed algorithms there is an increasing need for mathematical tools for analysis. Although there are several formalisms and tools for the analysis of ordinary distributed algorithms, there are not as many powerful tools for the analysis of randomization within distributed systems. This paper is part of a project that aims at developing the right math tools for proving properties of complicated randomized distributed algorithms and systems. The tools should be based on traditional probability theory, but at the same time should be tailored to the computational setting. Furthermore, the tools should have good facilities for modular reasoning due to the complexity of the systems to which they should be applied. The types of modularity we are looking for include parallel composition and abstraction mappings, but also anything else that decomposes the math analysis.

We develop our tools by analyzing complex algorithms of independent interest. In this paper we analyze the randomized consensus algorithm of Aspnes and Herlihy [3], which guarantees termination within expected polynomial time. The Aspnes-Herlihy algorithm is a rather complex algorithm. Processes move through a succession of asynchronous rounds, attempting to agree at each round. At each round, the agreement attempt involves a distributed random walk. The algorithm is hard to analyze because of its use of nontrivial results of probability theory (e.g., random walk theory), because of its complex setting, including asynchrony and both nondeterministic and probabilistic choice, and because of the interplay among several different sub-protocols.

* Supported by AFOSR-ONR contract F49620-94-1-0199, by ARPA contracts N00014-92-J-4033 and F19628-95-C-0118, and by NSF grant 9225124-CCR.

We formalize the Aspnes-Herlihy algorithm using probabilistic I/O automata [11]. In doing so, we decompose it formally into three subprotocols: one to carry out the agreement attempts, one to conduct the random walks, and one to implement a shared counter needed by the random walks. Properties of all three subprotocols are proved separately, and combined using general results about automaton composition. It turns out that most of the work involves proving non-probabilistic properties (invariants, simulation mappings, non-probabilistic progress properties, etc.). The probabilistic reasoning is isolated to a few small sections of the proof.

The task of carrying out this proof has led us to develop several general proof techniques for probabilistic I/O automata. These include ways to combine expectations for different complexity measures, to compose expected complexity properties, to convert probabilistic claims to deterministic claims, to use abstraction mappings to prove probabilistic properties, and to apply random walk theory in a distributed computational setting.

Previous work on verification of randomized distributed algorithms includes [8], where the randomized dining philosophers algorithm of [5] is shown to guarantee progress with probability 1, [6, 9], where the algorithm of [5] is shown to guarantee progress within expected constant time, and [1], where the randomized self-stabilizing minimum spanning tree algorithm of [2] is shown to guarantee stabilization within an expected time proportional to the diameter of a network. The analysis of [8] is based on converting a probabilistic property into a property of some of the computations of an algorithm (extreme fair computations); the analysis of [6, 9, 1] is based on part of the methodology used in this paper.

The paper is organized as follows. Section 2 presents the basic theoretical tools for our analysis; Section 3 presents the algorithm of Aspnes and Herlihy, describes the module that carries out the agreement attempts, and proves safety and liveness properties that do not depend on the details of the other modules; Section 5 builds the module that conducts the random walk and proves termination; Section 6 studies the expected time complexity of the algorithm; Section 7 gives some concluding remarks. In the presentation we focus mainly on the integration of probability with nondeterminism and we omit most of the analysis that does not involve probability.

2 Formal Model and Tools

2.1 Probabilistic I/O Automata

A *probability space* $\mathcal{P}$ 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 $\Omega \in \mathcal{F}$, 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 $C \subseteq \Omega$, $P[C] = \sum_{x \in C} P[\{x\}]$. For any arbitrary set X, let $Probs(X)$ denote the set of discrete probability spaces $(\Omega, \mathcal{F}, P)$ where $\Omega \subseteq X$, and such that all the elements of Ω have a non-zero probability.

An *I/O automaton* A consists of five components: a set $States(A)$ of states; a non-empty set $Start(A) \subseteq States(A)$ of start states; an action signature $Sig(A) = (in(A), out(A), int(A))$, where $in(A), out(A)$ and $int(A)$ are disjoint sets of input, output, and internal actions, respectively; a transition relation $Trans(A) \subseteq States(A) \times Actions(A) \times States(A)$, where $Actions(A)$ denotes the set $in(A) \cup out(A) \cup int(A)$, such that for each state s of $States(A)$ and each input action a of $in(A)$ there is a state s' such that $(s, a, s') \in Trans(A)$; a task partition $Tasks(A)$, which is an equivalence relation on $int(A) \cup out(A)$ that has at most countably many equivalence classes. The elements of $Trans(A)$ are called *transitions*, and A is said to be *input enabled*. An equivalence class of $Tasks(A)$ is called a *task* of A. A *probabilistic I/O automaton* M differs from an I/O automaton in its transition relation. That is, $Trans(M) \subseteq States(M) \times Actions(M) \times Probs(States(M))$. In the rest of the paper we refer to (probabilistic) I/O automata as (probabilistic) automata.

A state s of M is said to *enable* a transition if there is a transition $(s, a, \mathcal{P})$ in $Trans(M)$, and an action a is said to be enabled from s if s enables a transition with action a. 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, $\alpha = s_0 a_1 s_1 a_2 s_2 \ldots$, such that for each $i \geq 0$ there exists a transition $(s_i, a_{i+1}, \mathcal{P})$ of M such that $s_{i+1} \in \Omega$. Denote by $fstate(\alpha)$ the first state of α and, if α is finite, denote by $lstate(\alpha)$ the last state of α. An *execution* is an execution fragment whose first state is a start state. An execution fragment α is said to be *fair* iff the following conditions hold for every task T of M: 1) if α is finite then no action from T is enabled in $lstate(\alpha)$; 2) if α is infinite, then either actions from T occur infinitely many times in α, or α contains infinitely many occurrences of states from which no action from T is enabled. A state s of M is *reachable* if there exists a finite execution of M that ends in s. 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 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 $\alpha_1 \leq \alpha_2$, iff either $\alpha_1 = \alpha_2$ or α_1 is finite and there exists an execution fragment α_1' of M such that $\alpha_2 = \alpha_1 \frown \alpha_1'$.

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 structure similar to a cycle-free Markov chain, which we call a *probabilistic execution fragment* of M. From the point of view of the study of algorithms, the nondeterminism is resolved by an *adversary* that chooses a transition to schedule based on the past history of the system. A probabilistic execution is the result of the action of some adversary. A probabilistic execution can be thought of as the result of unfolding the transition relation of a probabilistic automaton and then choosing one transition for each state of the unfolding. It has a structure similar to the structure of a probabilistic automaton, where the states are finite execution fragments of M. It is possible to define a probability space $\mathcal{P}_H = (\Omega_H, \mathcal{F}_H, P_H)$ associated with H. In particular Ω_H is a set of execution fragments of M, $\mathcal{F}_H$ is the smallest σ-field that contains the set of *cones* C_q, consisting of those elements of Ω_H having q as a prefix (let q denote a state of H), and the probability measure P_H is the unique extension of the probability measure defined on cones as follows:

$P_H[C_q]$ is the product of the probabilities of each transition of H leading to q. An *event* E of H is an element of $\mathcal{F}_H$. An event E is called *finitely satisfiable* if it can be expressed as a union of cones. A finitely satisfiable event can be represented by a set Θ of incomparable states of H. The event denoted by Θ is $\cup_{q\in\Theta} C_q$. We abuse notation by writing $P_H[\Theta]$ for $P_H[\cup_{q\in\Theta} C_q]$. We call a set of incomparable states of H a *cut* of H, and we say that a cut Θ is *full* if $P_H[\Theta] = 1$. An important event of $\mathcal{P}_H$ is the set of fair executions of Ω_H. We define a probabilistic execution fragment H to be fair if the set of fair execution fragments has probability 1 in $\mathcal{P}_H$.

Probabilistic automata can be composed in parallel. The states of the composition are the cross product of the states of the components. The composed probabilistic automata synchronize on their common actions and evolve independently on the others. Whenever a synchronization occurs, the state that is reached is obtained by choosing a state independently for each of the probabilistic automata involved. In a parallel composition the notion of *projection* is one of the main tools to support modular reasoning. A projection of an execution fragment α onto a component within a parallel composition is the contribution of the component to obtain α. Formally, let M be $M_1 \parallel M_2$, the parallel composition of M_1 and M_2, and let α be an execution fragment of M. The projection of α onto M_i, denoted by $\alpha\lceil M_i$, is the sequence obtained from α by replacing each state with its i^{th} component and by removing all actions that are not actions of M_i together with their following state. It is the case that $\alpha\lceil M_i$ is an execution fragment of M_i. A similar construction is possible on probabilistic execution fragments. Here we just claim that $H\lceil M_i$ is a probabilistic execution fragment of M_i and that the probability space associated with $H\lceil M_i$ is the image space under projection of the probability space associated with H.

Proposition1 [10]. *Let M be $M_1 \parallel M_2$, and let H be a probabilistic execution fragment of M. Let $i \in \{1,2\}$. Then $\Omega_{H\lceil M_i} = \{\alpha\lceil M_i \mid \alpha \in \Omega_H\}$, and for each $\Theta \in \mathcal{F}_{H\lceil M_i}$, $P_{H\lceil M_i}[\Theta] = P_H[\{\alpha \in \Omega_H \mid \alpha\lceil M_i \in \Theta\}]$.*

2.2 Complexity Measures

A *complexity function* is a function from execution fragments of M to $\Re^{\geq 0}$. A *complexity measure* is a complexity function ϕ such that, for each pair α_1 and α_2 of execution fragments that can be concatenated, $max(\phi(\alpha_1), \phi(\alpha_2)) \leq \phi(\alpha_1 \frown \alpha_2) \leq \phi(\alpha_1) + \phi(\alpha_2)$. Informally, a complexity measure is a function that determines the complexity of an execution fragment. A complexity measure satisfies two natural requirements: the complexity of two tasks performed sequentially should not exceed the complexity of performing the two tasks separately and should be at least as large as the complexity of the more complex task.

Consider a probabilistic execution fragment H of M and a finitely satisfiable event Θ of $\mathcal{F}_H$. The elements of Θ represent the points where the property denoted by Θ is satisfied. Let ϕ be a complexity function. Then, define the expected complexity ϕ to reach Θ in H as

$$E_\phi[H,\Theta] \triangleq \begin{cases} \sum_{q\in\Theta} \phi(q) P_H[C_q] & \text{if } P_H[\Theta] = 1 \\ \infty & \text{otherwise.} \end{cases}$$

If several complexity measures are related by a linear inequality, then their expected values over a full cut are related by the same linear inequality. We use this result in the time analysis of the algorithm of Aspnes and Herlihy, where we express the time complexity of the protocol in terms of two other complexity measures.

Proposition 2. *Let Θ be a full cut of a probabilistic execution fragment H. Let ϕ, ϕ_1, ϕ_2 be complexity functions, and c_1, c_2 two constants such that, for each $\alpha \in \Theta$, $\phi(\alpha) \leq c_1\phi_1(\alpha) + c_2\phi_2(\alpha)$. Then $E_\phi[H, \Theta] \leq c_1 E_{\phi_1}[H, \Theta] + c_2 E_{\phi_2}[H, \Theta]$.*

Suppose now that within a computation it is possible to identify several phases, each one with its own complexity, and suppose that the complexity associated with each phase remains 0 until the phase starts. Suppose that the expected complexity of each phase is bounded by some constant c. If we know that the expected number of phases that start is bounded by k, then the expected complexity of the system is bounded by ck. The algorithm of Aspnes and Herlihy works in *rounds*, and at each round a special *coin flipping* protocol is run. The rounds can be seen as phases. The main difficulty is that several rounds may run concurrently.

Proposition 3. *Let M be a probabilistic automaton. Let $\phi_1, \phi_2, \phi_3, \ldots$ be a countable collection of complexity functions for M, and let ϕ' be a complexity function defined as $\phi'(\alpha) = \sum_{i \geq 0} \phi_i(\alpha)$. Let c be a constant, and suppose that for each fair probabilistic execution fragment H of M, each full cut Θ of H, and each $i > 0$, $E_{\phi_i}[H, \Theta] \leq c$.*

Let H be a probabilistic fair execution fragment of M, and let ϕ be a complexity measure for M. For each $i > 0$, let Θ_i be the set of minimal states q of H such that $\phi(q) \geq i$. Suppose that for each $q \in \Theta_i$, $\phi_i(q) = 0$, and that for each state q of H and each $i > \phi(q)$, $\phi_i(q) = 0$.

Then, for each full cut Θ of H, $E_{\phi'}[H, \Theta] \leq cE_\phi[H, \Theta]$.

Finally, to verify properties modularly it is useful to derive complexity properties of complex systems based on complexity properties of their components.

Proposition 4. *Let M be $M_1 \| M_2$, and let $i \in \{1, 2\}$. Let ϕ be a complexity function for M, and let ϕ_i be a complexity function for M_i. Suppose that for each finite execution fragment α of M, $\phi(\alpha) = \phi_i(\alpha \lceil M_i)$. Let c be a constant. Suppose that for each probabilistic execution fragment H of M_i and each full cut Θ of H, $E_{\phi_i}[H, \Theta] \leq c$. Then, for each probabilistic execution fragment H of M and each full cut Θ of H, $E_\phi[H, \Theta] \leq c$.*

2.3 Probabilistic Complexity Statements

A probabilistic complexity statement [9, 11] is a predicate that states whether all the fair probabilistic executions of a probabilistic automaton guarantee some reachability property within some complexity c with some minimum probability p. Probabilistic

complexity statements essentially express partial progress properties of a probabilistic system. Such partial progress properties can then be used to derive upper bounds on the expected complexity for progress. Formally, $U \xrightarrow[p]{\phi \leq c} U'$ is a predicate that is true for M iff for each fair probabilistic execution fragment H of M that starts from a state of U, $P_H[e_{U',\phi(c)}(H)] \geq p$, where $e_{U',\phi(c)}(H)$ denotes the set of executions α of Ω_H with a prefix α' such that $\phi(\alpha') \leq c$ and $lstate(\alpha') \in U'$.

Denote by $U \Rightarrow U \, unless \, U'$ the predicate that is true for M iff for every execution fragment sas' of M, $s \in U - U' \Rightarrow s' \in U \cup U'$. Informally, $U \Rightarrow U \, unless \, U'$ means that, once a state from U is reached, M remains in U unless U' is reached. For each probabilistic execution fragment H of M, let $\Theta_{U'}(H)$ denote the set of minimal states of H where a state from U' is reached. The following theorem provides a way of computing the expected ϕ for reaching U'.

Proposition 5 [11]. *Let M be a probabilistic automaton and ϕ be a complexity measure. Suppose that for each execution fragment of M of the form sas', $\phi(sas') \leq 1$, that is, each transition of M increases ϕ by at most 1. Let U and U' be sets of states of M. Let H be a probabilistic execution fragment of M that starts from a state of U, and suppose that for each state q of H such that $lstate(q) \in U$ some transition is scheduled with probability 1. Suppose also that $U \xrightarrow[p]{\phi \leq c} U'$ and $U \Rightarrow U \, unless \, U'$. Then, $E_\phi[H, \Theta_{U'}(H)] \leq (c+1)/p$.*

A useful technique to prove the validity of a probabilistic complexity statement $U \xrightarrow[p]{\phi \leq c} U'$ for a probabilistic automaton M is the following [9]: 1) choose a set of random draws that may occur within a probabilistic execution of M, and choose some of the possible outcomes; 2) show that, 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 random draws give the chosen outcome, a state from U' is reached within c units of complexity ϕ.

The first two steps can be carried out using the so-called *coin lemmas* [6, 9, 11], which provide rules to map a stochastic process onto a probabilistic execution and lower bounds on the probability of the mapped events based on the properties of the given stochastic process; the third step concerns non-probabilistic properties and can be carried out by means of any known technique for non-probabilistic systems. Coin lemmas are essentially a way of reducing the analysis of a probabilistic property to the analysis of an ordinary nondeterministic property.

2.4 Symmetric Random Walks for Probabilistic Automata

The correctness of the protocol of Aspnes and Herlihy is based on the theory of random walks [4]. That is, some parts of the protocol behave like a probabilistic process called random walk. The problem is to make sure that the protocol indeed behaves like a random walk. This is a point where intuition often fails, and therefore we need a proof technique that is sufficiently rigorous and simple to avoid mistakes.

Roughly speaking, a random walk is a process that describes the moves of a particle on the real line, where at each time the particle moves in one direction with probability p and in the opposite direction with probability $(1-p)$. In this section we present a coin lemma for symmetric random walks. That is, $p = 1/2$.

Let M be a probabilistic automaton and let $Acts = \{flip_1, \ldots, flip_n\}$ be a subset of $Actions(M)$. Let $\mathbf{S} = \{(U_1^h, U_1^t), (U_2^h, U_2^t), \ldots, (U_n^h, U_n^t)\}$ be a set of pairs where for each $i, 1 \leq i \leq n$, U_i^h, U_i^t are disjoint subsets of $States(M)$ such that for every transition $(s, flip_i, \mathcal{P})$ with an action $flip_i$, $\Omega \subseteq U_i^h \cup U_i^t$, and $P[U_i^h] = P[U_i^t] = 1/2$. The actions from $Acts$ represent coin flips, and the sets of states U_i^h and U_i^t represent the two possible outcomes. Given a finite execution fragment α of M, let $Diff_{Acts,\mathbf{S}}(\alpha)$ denote the difference between the heads and the tails that occur in H. Let z, B, and T be natural numbers, and let $B < T$. The value of z denotes the starting point of the particle, while B and T denote *barriers* in the real line. For each finite execution fragment α, let $z + Diff(\alpha)$ denote the position of the particle after the occurrence of α. For each probabilistic execution fragment H of M, let $\mathbf{Top}[B, T, z](H)$ be the set of executions α of Ω_H such that either the particle reaches the top barrier T before the bottom barrier B, or the total number of "flips" is finite and the particle reaches neither barrier. Define the symmetric event $\mathbf{Bot}[B, T, z](H)$, which is the same as **Top** except that the bottom barrier B should be reached before the top barrier T. Finally, define the event $\mathbf{Either}[B, T, z](H)$ as $\mathbf{Top}[B, T, z](H) \cup \mathbf{Bot}[B, T, z](H)$, which excludes those executions of M where infinitely many "flips" occur and the particle reaches neither barrier.

Proposition 6. *Let H be a probabilistic execution fragment of M, and let $B \leq z \leq T$. Then*

1. $P_H[\mathbf{Top}[B, T, z](H)] \geq (z - B)/(T - B)$.
2. $P_H[\mathbf{Bot}[B, T, z](H)] \geq (T - z)/(T - B)$.
3. $P_H[\mathbf{Either}[B, T, z](H)] = 1$.

We conclude with a result about the expected complexity of a random walk. Let $\phi_{Acts}(\alpha)$ be the complexity measure that counts the number of actions from $Acts$ that occur in α. Define $\phi_{Acts,B,T,z}$ to be the truncation of ϕ_{Acts} at the point where one of the barriers B and T is reached. Then we can prove an upper bound on the number of expected flip actions that occur before reaching one of the barriers.

Proposition 7. *Let H be a probabilistic execution fragment of M, and let Θ be a full cut of H. Let $B \leq z \leq T$. Then, $E_{\phi_{Acts,B,T,z}}[H, \Theta] \leq -z^2 + (B + T)z - BT$.*

3 The Algorithm of Aspnes and Herlihy

3.1 Description of the Algorithm

The algorithm of Aspnes and Herlihy proceeds in rounds. Every process maintains a variable with two fields, *value* and *round*, that contain the process' current preferred

value ($0, 1$ or $\perp$) and current round (a non-negative integer), respectively. We say that a process is at round r if its *round* field is equal to r. The variables (*value*, *round*) are multiple-reader single-writer. Each process starts with its *round* field initialized to 0 and its *value* field initialized to $\perp$.

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

We represent the main part of the algorithm as an automaton AP (Agreement Protocol) and the coin flipping protocols as probabilistic automata CF_r (Coin Flipper), one for each round r. With this decomposition we can analyze several properties just on AP using ordinary techniques for non-probabilistic systems. Indeed, in this section we deal with AP only, and we leave the coin flippers unspecified.

The formal definition of AP is given in Table 1. Beside the shared variables *value*(i) and i), each process has a program counter *pc*, two arrays *values* and *rounds* containing the scans of the other processes, a set variable *obs* saying what processes have been observed, a variable *start* holding the initial preferred value, and two variables *decided*, and *stopped* stating whether the process has decided or failed. We explain some of the relevant predicates: *obs-leader*(j) is true if i observes that j is a leader; *obs-agree*(r, v) is true if the observations of all the processes whose round is at least r agree on v; *obs-leader-agree*(v) is true if i observes that the leaders agree on a value v; *obs-leader-value* is the value of one of the leaders observed by i. We say that a process is *active* if it is attempting to agree on a value. An active process becomes inactive either by deciding a value or by failing.

3.2 Safety Properties

Validity states that "if a process decides on a value, then this value is the initial value of some process". The proof of validity derives from a trivial invariant saying that no process will ever prefer a value different from its initial value if all processes have the same initial value. Agreement states that "any two processes that decide within an execution of the algorithm decide on the same value". The key idea of the proof of agreement is that if a process i that is at round r is "about to decide" on some value v, then every process that is at round r or higher has its value equal to v. Define *agree*(r, v) to be true if all the processes at round at least r prefer v.

Actions and transitions of process i.

input $init(v)_i$
  Eff: $start \leftarrow v$

output $start(v)_i$
  Pre: $pc = init \wedge start = v \neq \perp$
  Eff: $value(i) \leftarrow v$
    $round(i) \leftarrow 1$
    $obs \leftarrow \emptyset$
    $pc \leftarrow read1$

output $read1(k)_i$
  Pre: $pc = read1$
    $k \notin obs$
  Eff: $values[k] \leftarrow value(k)$
    $rounds[k] \leftarrow round(k)$
    $obs \leftarrow obs \cup \{k\}$
    if $obs = \{1, \ldots, n\}$ then $pc \leftarrow check1$

output $check1_i$
  Pre: $pc = check1$
  Eff: if $\exists_{v \in \{0,1\}} obs\text{-}agree(rounds[i] - 1, v) \wedge$
      $obs\text{-}leader(i)$ then
      $pc \leftarrow decide$
    elseif $\exists_{v \in \{0,1\}} obs\text{-}leader\text{-}agree(v)$ then
      $value(i) \leftarrow obs\text{-}leader\text{-}value$
      $round(i) \leftarrow rounds[i] + 1$
      $obs \leftarrow \emptyset$
      $pc \leftarrow read1$
    else
      $value(i) \leftarrow \perp$
      $obs \leftarrow \emptyset$
      $pc \leftarrow read2$

output $decide(v)_i$
  Pre: $pc = decide \wedge values[i] = v$
  Eff: $decided \leftarrow true$
    $pc \leftarrow nil$

output $read2(k)_i$
  Pre: $pc = read2$
    $k \notin obs$
  Eff: $values[k] \leftarrow value(k)$
    $rounds[k] \leftarrow round(k)$
    $obs \leftarrow obs \cup \{k\}$
    if $obs = \{1, \ldots, n\}$ then
      $pc \leftarrow check2$

output $check2_i$
  Pre: $pc = check2$
  Eff: if $\exists_{v \in \{0,1\}} obs\text{-}leader\text{-}agree(v)$ then
      $value(i) \leftarrow obs\text{-}leader\text{-}value$
      $round(i) \leftarrow rounds[i] + 1$
      $obs \leftarrow \emptyset$
      $pc \leftarrow read1$
    else
      $pc \leftarrow flip$

output $start\text{-}flip(r)_i$
  Pre: $pc = flip$
    $round(i) = r$
  Eff: $pc \leftarrow wait$

input $return\text{-}flip(v, r)_i$
  Eff: if $pc = wait \wedge round(i) = r$ then
      $value(i) \leftarrow v$
      $round(i) \leftarrow rounds[i] + 1$
      $obs \leftarrow \emptyset$
      $pc \leftarrow read1$

input $stop_i$
  Eff: $stopped \leftarrow true$
    $pc \leftarrow nil$

Tasks: The locally controlled actions of process i form a single task.

Table 1. The actions and transition relation of AP.

Invariant 4 *Given a reachable state of AP, let $v = value(i)$ and $r = round(i)$. Then $(obs\text{-}agree(r-1,v)_i \wedge obs\text{-}leader(i)_i \wedge obs_i = \{1,\ldots,n\}) \Rightarrow agree(r,v)$.*

Invariant 4 is sufficient to prove agreement. The idea is that the premise of Invariant 4 is stable. In fact, if process i satisfies the premise of Invariant 4, then process i decides on value v, and thus the local state of process i does not change any more. The analysis of Invariant 4 follows standard methods for invariant proofs within ordinary nondeterministic systems, and is based on several other invariants. The main invariant, which we omit here, is expressed in a new style that we think is useful: it talks about the state of a process when it is in the middle of a scanning pass, and describes properties that would hold if the scanning pass is completed instantly.

4.1 Non-Probabilistic Progress Properties

Our next objective is to show that in the algorithm of Aspnes and Herlihy some decision is reached within some expected number of rounds. This property depends on the probabilistic properties of the coin flipping protocols. However, there are several progress properties of the algorithm that do not depend on any probabilistic assumption. In this section we study such properties. The advantage of this approach is that we can use most of the existing techniques for ordinary nondeterministic systems and confine probabilistic arguments to a limited section of the analysis.

For each round r, let CF_r denote the coin flipping protocol for round r. Define AH to be $AP \parallel (\parallel_{r \geq 1} CF_r)$. For each finite execution fragment α of AH, define the complexity measure $\phi_{MaxRound}(\alpha)$ as the difference between the maximum round numbers of the final and initial states of α. Define the following sets of states.

$\mathcal{R}$ the set of reachable states of AH such that there is an active process;
$\mathcal{D}$ the set of reachable states of AH such that there is no active process.

We show that, under some conditions on the coin flipping protocols, starting from any state of $\mathcal{R}$, a state from $\mathcal{D}$ is reached within some bounded number of rounds. We split the problem: first we show that, unless the algorithm terminates, the system reaches a point where one process just moved to a new maximum round; then, we show that from such an intermediate point the algorithm terminates. The proofs are based on simple invariants. Formally, for $v \in \{0,1\}$, define the following set of states.

$\mathcal{F}_v$ the set of states of $\mathcal{R}$ where there exists a round r and a process l such that $round(l) = r$, $value(l) = v$, $obs_l = \emptyset$, and for all processes $j \neq l$, $round(j) < r$.

Proposition 8. *If AH is in a state s of $\mathcal{R}$ and all invocations to the coin flippers on non-failing ports get a response, then a state from $\mathcal{F}_0 \cup \mathcal{F}_1 \cup \mathcal{D}$ is reached within one round.*

Proposition 9. *If AH is in a state s of $\mathcal{F}_v$, all invocations to the coin flippers on non-failing ports get a response, and all invocations to $CF_{s.max\text{-}round}$ get only response v, then a state from $\mathcal{D}$ is reached within two rounds.*

4.2 Probabilistic Progress Properties

Suppose that each coin flipping protocol CF_r satisfies the following properties.

C1 For each fair probabilistic execution fragment of CF_r that starts with a reachable state of CF_r, the probability that each invocation on a non-failing port gets a response is 1.

C2 For each fair probabilistic execution of CF_r, and each value $v \in \{0,1\}$, the probability that all invocations on a non-failing port get response v is at least p, $0 < p \leq 1$.

Proposition 10. *If each coin flipping protocol CF_r satisfies properties* **C1** *and* **C2**, *then in AH, starting from any state of $\mathcal{R}$ and under any fair scheduler, a state from $\mathcal{D}$ is reached within $O(1/p)$ expected rounds.*

Proof. We first derive the statement $\mathcal{R} \xrightarrow[p]{\phi_{MaxRound} \leq 3} \mathcal{D}$ from two intermediate statements $\mathcal{R} \xrightarrow[1]{\phi_{MaxRound} \leq 1} \mathcal{F}_0 \cup \mathcal{F}_1 \cup \mathcal{D}$ and $\mathcal{F}_v \xrightarrow[p]{\phi_{MaxRound} \leq 2} \mathcal{D}$, $v \in \{0,1\}$. The proofs of the intermediate statements rely on Propositions 8 and 9 and on **C1** and **C2**. Since in AH $\mathcal{R}$ is not left unless a state from $\mathcal{D}$ is reached, since each transition of AH increases $\phi_{MaxRound}$ by at most 1, and since from fairness and **C1** some transition is scheduled with probability 1 from each state of $\mathcal{R}$, by Theorem 5 we derive that within expected $4/p$ rounds a state from $\mathcal{D}$ is reached under any fair scheduler.

5 The Atomic Coin Flipping Protocol

5.1 The Protocol

We build a coin flipping protocol that satisfies **C1** and **C2** with $p = (K-1)/2K$. The protocol is based on random walks. We define the protocol by letting a probabilistic automaton DCN_r (Distributed CoiN) interact with a non-probabilistic counter CT_r (CounTer), that is, $CF_r = DCN_r \parallel CT_r$. In this Section, DCN_r is distributed while CT_r is composed of n processes that receive requests from DCN_r and read/update a single shared variable. In Section 5.4 we discuss how to decentralize CT_r. Since the protocols for DCN_r and CT_r are the same for any round r, we drop the subscript r from our notation. In DCN each process flips a fair coin to decide whether to increment or decrement the shared counter. Then the process reads the current value of the shared counter by invoking CT, and if the value read is beyond the barrier $-Kn$ $(+Kn)$, where K is a fixed constant, then the process returns 0 (1). The specification of CT states that an increment or decrement operation always completes unless the corresponding process fails, while a read operation is guaranteed to complete only if increments and decrements eventually cease.

5.2 Non-Probabilistic Analysis

Let $Acts$ be $\{flip_1, \ldots, flip_n\}$, and let $\mathbf{S}$ be $\{(U_1^i, U_1^d), (U_2^i, U_2^d), \ldots, (U_n^i, U_n^d)\}$, where U_j^i is the set of states of CF where process j has just flipped inc ($fpc_j = inc$), and U_j^d is the set of states of CF where process j has just flipped dec ($fpc_j = dec$). Given a finite execution fragment α of CF, let $\phi_{inc}(\alpha)$ be the number of coin flips in α that give inc, and let $\phi_{dec}(\alpha)$ be the number of coin flips in α that give dec.

Lemma 11. *Let α be a fair execution of CF, such that $\alpha \in \mathbf{Top}[-(K-1)n, (K+1)n, 0](H)$ for some probabilistic execution H of CF. Then in α every invocation on a non-failing port gets response 1.*

The proof of Lemma 11 follows from simple invariant properties. The main idea is that the value of the shared counter remains beyond Kn once the barrier $(K+1)n$ is reached. A symmetric argument is valid for $\mathbf{Bottom}[-(K-1)n, (K+1)n, 0](H)$.

5.3 Probabilistic Analysis

We prove only **C2** by applying our coin lemma for random walks.

Proposition 12. *The coin flipper CF satisfies* **C2** *with $p = (K+1)/2K$. That is, fixed $v \in \{0,1\}$, for each fair probabilistic execution of CF, with probability at least $(K-1)/2K$ each invocation to CF on a non-failing port returns value v.*

Proof. Assume that $v = 1$; the case for $v = 0$ is symmetric. Let H be a fair probabilistic execution of CF. If α is an execution of $\mathbf{Top}[-(K-1)n, (K+1)n, 0](H)$, then, by Lemma 11, every invocation to CF in α gets response 1. Furthermore, by Theorem 6, $P_H[\mathbf{Top}[-(K-1)n, (K+1)n, 0](H)] \geq (K-1)/2K$.

5.4 Implementation of the Shared Counter

It is possible to build a distributed implementation of CT that preserves **C1** and **C2**. The implementation, which we denote by DCT (Distributed CounTer), is presented in [3]. In the full paper we verify that DCT implements CT by exhibiting a *refinement mapping* [7] from DCT to CT. This part of the proof is simple and does not involve probability. Then we use the compositionality results of [11] to show that DCT can replace CT in AH.

5.5 Summing Up

In this section we paste together the results of the previous sections to derive an upper bound on the expected number of rounds for termination. In particular, if we know that there is at least one initialized process that does not fail, then we know that a decision is reached within constant many rounds.

Theorem 13. *Using either the counters CT or DCT, from each reachable state of AH, under any fair scheduler, a state of $\mathcal{D}$ is reached within a constant expected number of rounds.*

Proof. The coin flippers with the counters CT or DCT satisfy properties **C1** and **C2** with $p = (K-1)/2K$. By Proposition 10, AH guarantees that $\mathcal{D}$ is reached within at most $O(2K/(K-1))$ expected rounds.

6 Timing Analysis of the Algorithm

In this section we derive an upper bound on the time to reach $\mathcal{D}$ once all processes have some minimum speed. We achieve this result by studying the expected number of *inc* and *dec* events that occur within the coin flippers and then converting the new expected bound into a time bound.

We change slightly our formal model to handle time. Specifically, we add a component $.now$ to the states of all our probabilistic I/O automata, and we add the set of positive real numbers to the input actions of all our probabilistic I/O automata. The $.now$ component is a nonnegative real number and describes the current time of an automaton. At the beginning (i.e., in the start states) the current time is 0, and thus the $.now$ component is 0. The occurrence of an action d, where d is a positive real number, increments the $.now$ component by d and leaves the rest of the state unchanged. Thus, the occurrence of an action d models the fact that d time units are elapsing. The amount of time elapsed since the beginning of an execution is recorded in the $.now$ component. Since time-passage actions must synchronize in a parallel composition context, parallel composition ensures that the $.now$ components of the components are always equal. Thus, we can abuse notation and talk about the $.now$ component of the composition of two automata while we refer to the $.now$ component of one of the components. We define a new complexity measure $\phi_t(\alpha)$ as the difference between the $.now$ components of the last and first states of α. Informally, ϕ_t measures the time that elapses during an execution. We say that an execution fragment α of a probabilistic automaton M is *well-timed* if each task does not remain enabled for more than one time unit without being performed.

We give some preliminary definitions. Let, for each $r > 0$, DCF_r (Distributed Coin Flipper) denote $DCN_r \,||\, DCT_r$. Let DAH (Distributed Aspnes-Herlihy) denote $AP \,||\, (||_{r \geq 1} DCF_r)$. For an execution fragment α of DCF_r or of DAH, let $\phi_{flip,r}(\alpha)$ be the number of *flip* events of DCF_r that occur in α, and let $\phi_{id,r}(\alpha)$ be the number of *inc* and *dec* events of DCF_r that occur in α. For each execution fragment α of DAH let $\phi_{id}(\alpha)$ be the number of *inc* and *dec* events that occur in α.

We start with some non-probabilistic properties about the new complexity measures. The first result, Lemma 14, provides a linear upper bound on the time it takes for DAH to span a given number of rounds and to flip a given number of coins under the assumption of well-timedness. The next two results state basic properties of the coin flipping protocols. That is, once a barrier $\pm(K+1)n$ is reached, there are at most n other *flip* events, and within any execution fragment of DCF_r the difference between the *inc*, *dec* events and the *flip* events is at most n.

Lemma 14. *Let α be a well-timed execution fragment of DAH, and suppose that all the states of α, with the possible exception of lstate(α) are active, that is, are states of $\mathcal{R}$. Let $R = fstate(\alpha).max\text{-}round$. Then, $\phi_t(\alpha) \leq d_1 n^2(\phi_{MaxRound}(\alpha) + R) + d_2 n \phi_{id}(\alpha) + d_3 n^2$ for some constants d_1, d_2, and d_3.*

Lemma 15. *Let $\alpha = \alpha_1 \frown \alpha_2$ be a finite execution of DCF_r, and suppose that $|Diff_{Acts,\mathbf{S}}(\alpha_1)| \geq (K+1)n$. Then $\phi_{flip,r}(\alpha_2) \leq n$.*

Lemma 16. *Let α be a finite execution fragment of DCF_r. Then, $\phi_{id,r}(\alpha) \leq \phi_{flip,r}(\alpha) + n$.*

We now deal with probabilistic properties. First, based on our results on random walks and on Lemma 15, we show in Lemma 17 an upper bound on the expected number of coin flips performed by a coin flipper. Then, in Lemma 18 we use Lemma 16 and our results about linear combinations of complexity measures to derive an upper bound on the expected number of increment and decrement operations performed by a coin flipper, and we use our compositionality result about complexity measures to show that the bound is preserved by parallel composition. Finally, in Lemma 19 we use our result about phases of computations to combine Theorem 13 with Lemma 18 and derive an upper bound on the expected number of *inc* and *dec* events performed by the algorithm.

Lemma 17. *Let H be a probabilistic execution fragment of DCF_r that starts from a reachable state, and let Θ be a full cut of H. Then $E_{\phi_{flip,r}}[H, \Theta] \leq (K+1)^2 n^2 + n$.*

Lemma 18. *Let H be a probabilistic execution fragment of DAH that starts from a reachable state, and let Θ be a full cut of H. Then $E_{\phi_{id,r}}[H, \Theta] \leq (K+1)^2 n^2 + 2n$.*

Lemma 19. *Let H be a probabilistic fair execution fragment of DAH with start state s, and let $R = s.max\text{-}round$. Suppose that s is reachable. Let Θ denote the set of minimal states of H where a state from $\mathcal{D}$ is reached. Then $E_{\phi_{id}}[H, \Theta] = O(Rn^2)$.*

The main result is just a pasting together of the results obtained so far. An immediate consequence on the algorithm of Aspnes and Herlihy is that, if we know that some initialized process does not fail and that the maximum round is 1, then a decision is reached within expected cubic time.

Theorem 20. *Let H be a probabilistic fair, well-timed execution fragment of DAH with a reachable start state s, and let $R = s.max\text{-}round$. Let Θ denote the set of minimal states of H where a state from $\mathcal{D}$ is reached. Then $E_{\phi_t}[H, \Theta] = O(Rn^3)$.*

Proof. By Lemma 14 and Proposition 2, $E_{\phi_t}[H, \Theta] \leq d_1 n^2 E_{\phi_{MaxRound}}[H, \Theta] + d_1 n^2 R + d_2 n E_{\phi_{id}}[H, \Theta] + d_3 n^2$. Thus, by Theorem 13 and Lemma 19, $E_{\phi_t}[H, \Theta] = O(Rn^3)$.

7 Concluding Remarks

In the full paper [10] the length of the analysis of the Aspnes-Herlihy algorithm is double the length of the original proof of Aspnes and Herlihy [3]. This shows that it is possible to prove formally and rigorously the correctness of a randomized distributed algorithm without using too much space. Furthermore, even though in the full paper we have proved all the results, a shorter high level analysis of a protocol using our tools is sufficient to increase considerably our confidence in the correctness of the protocol. The high level analysis provides a designer with a collection of simple properties to check so that the possible subtleties of randomization can be discovered.

References

1. S. Aggarwal. Time optimal self-stabilizing spanning tree algorithms. Technical Report MIT/LCS/TR-632, MIT Laboratory for Computer Science, 1994. Master's thesis.
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*, volume 761 of *Lecture Notes in Computer Science*, pages 400–410, Bombay, India., December 1993. Springer-Verlag.
3. J. Aspnes and M.P. Herlihy. Fast randomized consensus using shared memory. *Journal of Algorithms*, 15(1):441–460, September 1990.
4. W. Feller. *An Introduction to Probability Theory and its Applications. Volume 1.* Jokn Wiley & Sons, Inc., 1950.
5. 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 Symposium on Principles of Programming Languages*, pages 133–138, January 1981.
6. N.A. Lynch, I. Saias, and R. Segala. Proving time bounds for randomized distributed algorithms. In *Proceedings of the* 13^{th} *Annual ACM Symposium on Principles of Distributed Computing,* Los Angeles, CA, pages 314–323, 1994.
7. Nancy Lynch and Frits Vaandrager. Forward and backward simulations – Part II: Timing-based systems. *Information and Computation*, 121(2):214–233, September 1995.
8. A. Pnueli and L. Zuck. Verification of multiprocess probabilistic protocols. *Distributed Computing*, 1(1):53–72, 1986.
9. A. Pogosyants and R. Segala. Formal verification of timed properties of randomized distributed algorithms. In *Proceedings of the* 14^{th} *Annual ACM Symposium on Principles of Distributed Computing,* Ottawa, Ontario, Canada, pages 174–183, August 1995.
10. A. Pogosyants, R. Segala, and N. Lynch. Verification of the randomized consensus algorithm of Aspnes and Herlihy: a case study. Technical Memo MIT/LCS/TM-555, MIT Laboratory for Computer Science, 1997.
11. R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems.* PhD thesis, MIT, Dept. of Electrical Engineering and Computer Science, 1995. Also appears as technical report MIT/LCS/TR-676.