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

Anna Pogosyants† Roberto Segala‡ Nancy Lynch*

April 25, 1997

Abstract

The Probabilistic I/O Automaton model of [16] is used as the basis for a formal presentation and proof of the randomized consensus algorithm of Aspnes and Herlihy. The algorithm 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 (specifically, 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.

We formalize the Aspnes-Herlihy algorithm using probabilistic I/O automata. 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. We apply all of these techniques to analyze the expected complexity of the algorithm.

This paper is written in memory of Anna Pogosyants, who died in a car crash in December 1995 while working on this project for her Ph.D. dissertation.

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

†Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139 USA, lynch@theory.lcs.mit.edu

‡Dipartimento di Scienze dell'Informazione, Università di Bologna, Piazza di Porta San Donato 5, 40127 Bologna - Italy, segala@cs.unibo.it

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 we want to develop 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 (specifically, 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.

We formalize the Aspnes-Herlihy algorithm using probabilistic I/O automata [16]. 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. We apply all of these techniques to analyze the expected complexity of the algorithm.

[[[**RS.** Added reference to Sonu's thesis. Removed claim about first attempt.]]] Previous work on verification of randomized distributed algorithms includes [14], where the randomized dining philosophers algorithm of [9] is shown to guarantee progress with probability 1, [11, 15], where the algorithm of [9] 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 [14] is based on converting a probabilistic property

into a property of some of the computations of an algorithm (extreme fair computations); the analysis of [11, 15, 1] is based on part of the methodology used in this paper. Other work is based on probabilistic model checking (e.g, [17, 7]).

[[[**RS.** *** Add here references to other papers on randomized consensus.]]]

The rest of the paper is organized as follows. Section 2 presents the basic theoretical tools for our analysis, including probabilistic I/O automata, abstract complexity measures, progress statements and refinement mappings; Section 3 presents a coin lemma for random walks and a result about the expected complexity of a random walk within a probabilistic I/O automaton; Section 4 presents the algorithm of Aspnes and Herlihy and describes formally the module that carries out the agreement attempts; Sections 5 and 6 prove that the Aspnes-Herlihy algorithm satisfies the validity and agreement properties; Section 7 proves several progress properties of the algorithm that are not based on any probabilistic argument; Section 8 proves the probabilistic progress properties of the algorithm by using the results of Section 7; Section 9 builds the module that conducts the random walk; Section 10 builds the shared counter needed in Section 9; Section 11 derives the termination properties of the algorithm, where the complexity is measured in terms of expected number of rounds; Section 12 studies the expected time complexity of the algorithm; Section 13 gives some concluding remarks and discusses the kinds of modularization that we use in the proof.

Part I: The Underlying Theory

2 Formal Model and Tools

In this section we introduce the formalism that we use in the paper. We start with ordinary I/O automata following the style of [12, 10]; then we move to probabilistic I/O automata by adding the input/output structure to the probabilistic automata of [16]. We describe methods to handle complexity measures within probabilistic automata, and we present progress statements as a basic tool for the complexity analysis of a probabilistic system. Finally, we describe verification techniques based on refinements and traces.

2.1 I/O Automata

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: $in(A)$ is the set of input actions, $out(A)$ is the set of output actions, and $int(A)$ is the set of internal actions.
- 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') is an element of $Trans(A)$. The elements of $Trans(A)$ are called *transitions*, and A is said to be *input enabled*.
- a task partition $Tasks(A)$, which is an equivalence relation on $int(A) \cup out(A)$ that has at most countably many equivalence classes. An equivalence class of $Tasks(A)$ is called a *task* of A.

In the rest of the paper we refer to I/O automata as automata.

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 $Trans(A)$; a task T of A is said to be *enabled* from s if there is an action $a \in T$ that is enabled from s.

An *execution fragment* of an 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, $\alpha = s_0 a_1 s_1 a_2 s_2 ...$, such that for each $i \geq 0$ there exists a transition (s_i, a_{i+1}, 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. 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 fragment α is said to be *fair* iff the following conditions hold for every task T of A:

1. if α is finite then T is not 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 T is not enabled.

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 property ϕ of states is said to be *stable* for any execution fragment $\alpha = s_0 a_1 s_1 \cdots$ if ϕ continues to be satisfied once it is satisfied. That is, $\phi(s_0) \Rightarrow \forall_{i \geq 0} \phi(s_i)$.

A finite execution fragment $\alpha_1 = s_0 a_1 s_1 \cdots a_n s_n$ of A and an execution fragment $\alpha_2 = s_n a_{n+1} s_{n+1} \cdots$ of A can be *concatenated*. The concatenation, written $\alpha_1 \frown \alpha_2$, is the execution 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 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 $\alpha \triangleright \alpha_1$.

2.2 Probabilistic I/O Automata

2.2.1 Preliminaries on Probability Theory

A *probability space* is a triplet $(\Omega, \mathcal{F}, P)$ where

1. Ω is a set, also called the *sample space*,
2. $\mathcal{F}$ 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
3. 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 $\mathcal{F}$, $P[\cup_i C_i] = \sum_i P[C_i]$.

The pair $(\Omega, \mathcal{F})$ is called a *measurable space*, and the measure P is called a *probability measure*.

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 distributions whose sample space is a subset of X and such that all the elements of the sample space have a non-zero probability.

A function $f : \Omega_1 \to \Omega_2$ is said to be *measurable* from $(\Omega_1, \mathcal{F}_1)$ to $(\Omega_2, \mathcal{F}_2)$ if for each $E \in \mathcal{F}_2$, $f^{-1}(E) \in \mathcal{F}_1$. Given a probability space $(\Omega_1, \mathcal{F}_1, \mathcal{P}_1)$, a measurable space $(\Omega_2, \mathcal{F}_2)$,

and a measurable function f from $(\Omega_1, \mathcal{F}_1)$ to $(\Omega_2, \mathcal{F}_2)$, let $f(P_1)$, the *image measure* of P_1, be the measure defined on $(\Omega_2, \mathcal{F}_2)$ as follows: for each $E \in \mathcal{F}_2$, $f(P_1)(E) = P_1(f^{-1}(E))$. Standard measure theory arguments show that $(\Omega_2, \mathcal{F}_2, P_2)$ is a probability space. If $(\Omega, \mathcal{F}, P)$ is discrete, then we can define $f((\Omega, \mathcal{F}, P))$ as $(f(\Omega), 2^{f(\Omega)}, f(P))$.

For notational convenience we denote a probability space $(\Omega, \mathcal{F}, P)$ by $\mathcal{P}$. We also use primes and indices that carry over automatically to the components of a probability space. Thus, for example, $\mathcal{P}'_i$ denotes $(\Omega'_i, \mathcal{F}'_i, P'_i)$.

Given a probability space $\mathcal{P}$ and a set X, we abuse notation and we write $P[X]$ even if X contains elements that are not in Ω. By writing $P[X]$ we mean implicitly $P[X \cap \Omega]$. Also, given an element x, we write $P[x]$ for $P[\{x\}]$.

Given two discrete probability spaces $\mathcal{P}_1$ and $\mathcal{P}_2$, define the product $\mathcal{P}_1 \otimes \mathcal{P}_2$ of $\mathcal{P}_1$ and $\mathcal{P}_2$ to be the triplet $(\Omega_1 \times \Omega_2, 2^{\Omega_1 \times \Omega_2}, P_1 \otimes P_2)$, where, for each $(x_1, x_2) \in \Omega_1 \times \Omega_2$, $P_1 \otimes P_2[(x_1, x_2)] = P_1[x_1]P_2[x_2]$.

We conclude with some notions about random variables that are needed in some of the proofs of our results. Let $(\Re, \mathcal{F}_\Re)$ be a measurable space with the real numbers as sample space. Given a probability space $\mathcal{P}$, a *random variable* X for $\mathcal{P}$ is a measurable function from $(\Omega, \mathcal{F})$ to $(\Re, \mathcal{F}_\Re)$. As an example, a random variable could be the function that expresses the complexity of each element of Ω. It is possible to study the *expected value* of a random variable, that is, the average complexity of the elements of Ω, as follows: $E[X] = \sum_{x \in \Omega} X(x)P[x]$. A useful property of expected values is the following.

Proposition 2.1 *Let $\mathcal{P}$ be a probability space and let X be a random variable for $\mathcal{P}$. For $i \geq 0$, let the expression $X \geq i$ denote the event $\{x \in \Omega \mid X(x) \geq i\}$.*

1. *If the range of X is the set of natural numbers, then $E[X] = \sum_{i>0} P[X \geq i]$.*
2. *$E[X] \geq \sum_{i>0} P[X \geq i]$.*

Proof. For $i \geq 0$, let the expression $X = i$ denote the event $\{x \in \Omega \mid X(x) = i\}$. If the range of X is the set of natural numbers, then the expression for $E[X]$ can be rewritten as $E[X] = \sum_{i>0} iP[X = i]$. That is, $E[X]$ is a sum of terms such that each term $P[X = i]$ appears i times. By rearranging the terms we obtain $E[X] = \sum_{i>0} \sum_{j \geq i} P[X = j]$, that is, $E[X] = \sum_{i>0} P[X \geq i]$. This proves the first item. For the second item, let $\overline{X}$ be defined as follows: for each $x \in \Omega$, $\overline{X}(x) = \lfloor X(x) \rfloor$. It is easy to show that $\overline{X}$ is a random variable. From the definition of $\overline{X}$, $E[X] \geq E[\overline{X}]$ and for each $i > 0$, $P[X \geq i] = P[\overline{X} \geq i]$. Thus, using item 1, $E[X] \geq E[\overline{X}] = \sum_{i>0} P[\overline{X} \geq i] = \sum_{i>0} P[X \geq i]$. ■

2.2.2 Probabilistic I/O Automata

A *probabilistic I/O automaton* M consists of five components:

- a set $States(M)$ of states.
- a non-empty set $Start(M) \subseteq States(M)$ of start states;
- an action signature $Sig(M)$.
- a transition relation $Trans(M) \subseteq States(M) \times Actions(M) \times Probs(States(M))$ such that for each state s of $States(M)$ and each input action a of $in(M)$ there is a distribution $\mathcal{P}$ such that $(s, a, \mathcal{P})$ is an element of $Trans(M)$. We say that M is *input-enabled*.
- a task partition $Tasks(M)$, which is an equivalence relation on $int(M) \cup out(M)$ that has at most countably many equivalence classes.

[[[**RS.** Added definition of input-enabled. Why should I emphasize that locally controlled actions lead to probability distributions?]]] In the rest of the paper we refer to probabilistic I/O automata as probabilistic automata.

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 $(s_i, a_{i+1}, \mathcal{P})$ 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.

2.2.3 Probabilistic Executions

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. We also allow an adversary to use randomization in its choices, that is, a transition to be chosen probabilistically. This models the fact that the environment of a probabilistic automaton may provide input randomly.

Formally, a *probabilistic execution fragment* H of a probabilistic automaton M consists of four components.

- a set of states $States(H) \subseteq frag^*(M)$; let q range over the states of H;
- a signature $Sig(H) = Sig(M)$;
- a singleton set $Start(H) \subseteq States(M)$;

- a transition relation $Trans(H) \subseteq States(H) \times Probs((Actions(H) \times States(H)) \cup \{\delta\})$ such that for each transition $(q, \mathcal{P})$ of H there is a family $\{(lstate(q), a_i, \mathcal{P}_i)\}_{i \geq 0}$ of transitions of M and a family $\{p_i\}_{i \geq 0}$ of probabilities satisfying the following properties: $\sum_{i \geq 0} p_i \leq 1$, $P[\delta] = 1 - \sum_{i \geq 0} p_i$, and for each action a and state s, $P[(a, qas)] = \sum_{i | a_i = a} p_i \bar{P}_i[s]$.

Furthermore, each state of H is reachable, where reachability is defined analogously to the notion of reachability for probabilistic automata after defining an execution of a probabilistic execution fragment in the obvious way. A *probabilistic execution* H of a probabilistic automaton M is a probabilistic execution fragment of M whose start state is a state of $Start(M)$.

A probabilistic execution is like a probabilistic automaton, except that within a transition it is possible to choose probabilistically over actions as well. Furthermore, a transition may contain a special symbol δ, which corresponds to not scheduling any transition. In particular, it is possible that from a state q a transition is scheduled only with some probability $p < 1$. In such a case the probability of δ is $1 - p$.

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 Ω_H is the limit closure of $States(H)$, where the limit is taken under prefix ordering. The σ-field $\mathcal{F}_H$ is the smallest σ-field that contains the set of *cones* C_q, consisting of those executions of Ω_H having q as a prefix. 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. It is possible to show 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. The proof is analogous to the proof given in [16] for a similar probability space.

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 of incomparable states of H, that is, by a set $\Theta \subseteq States(H)$ such that for each $q_1, q_2 \in \Theta$, $q_1 \not\leq q_2$ and $q_2 \not\leq q_1$. 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 $P_H[\Theta] = 1$. Denote by $cuts(H)$ the set of cuts of H, and denote by *full-cuts*(H) the set of full cuts of H.

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 executions has probability 1 in $\mathcal{P}_H$.

We conclude by extending the $\triangleright$ operator to probabilistic execution fragments. Given a probabilistic execution fragment H of M and a state q of H, define $H \triangleright q$ (the fragment of H given that q has occurred), to be the probabilistic execution fragment of M obtained from H by removing all the states that do not have q as a prefix, by replacing all other states q' with $q' \triangleright q$, and by defining $lstate(q)$ to be the new start state. An important property of $H \triangleright q$ is the following.

Proposition 2.2 *For each state q' of $H \triangleright q$, $P_{H \triangleright q}[C_{q'}] = P_H[C_{q^\frown q'}]/P_H[C_q]$.* ■

2.3 Parallel Composition

Two probabilistic automata M_1 and M_2 are *compatible* iff $int(M_1) \cap acts(M_2) = \emptyset$ and $acts(M_1) \cap int(M_2) = \emptyset$. The *parallel composition* of two compatible probabilistic automata M_1 and M_2, denoted by $M_1 \parallel M_2$, is the probabilistic automaton M such that

1. $States(M) = States(M_1) \times States(M_2)$.
2. $Start(M) = Start(M_1) \times Start(M_2)$.
3. $Sig(M) = ((in(M_1) \cup in(M_2)) - (out(M_1) \cup out(M_2)), (int(M_1) \cup int(M_2)), (out(M_1) \cup out(M_2)))$.
4. $((s_1, s_2), a, \mathcal{P}) \in Trans(M)$ iff $\mathcal{P} = \mathcal{P}_1 \otimes \mathcal{P}_2$ where
   (a) if $a \in Actions(M_1)$ then $(s_1, a, \mathcal{P}_1) \in Trans(M_1)$, else $\mathcal{P}_1 = \mathcal{U}(s_1)$, and
   (b) if $a \in Actions(M_2)$ then $(s_2, a, \mathcal{P}_2) \in Trans(M_2)$, else $\mathcal{P}_2 = \mathcal{U}(s_2)$,

where $\mathcal{U}(s)$ denotes a probability distribution over a single state s. Informally, two 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 in a parallel composition context is the contribution of the component to obtain α. Formally, let M be $M_1 \parallel M_2$, 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.

The notion of projection can be extended to probabilistic executions (cf. Section 4.3 of [16]). Here we do not present the formal definition of projection; rather, we present some properties of a projection that are needed for our analysis, and we refer the reader to [16] for a more detailed description. Given a probabilistic execution fragment H of M, it is possible to define an object $H \lceil M_i$, which is a probabilistic execution fragment of M_i that informally represents the contribution of M_i to H. The states of $H \lceil M_i$ are the projections onto M_i of the states of H. The most important fact is that the probability space associated with $H \lceil M_i$ is the image space under projection of the probability space associated with H. This property allows us to prove probabilistic properties of H based on probabilistic properties of $H \lceil M_i$. [[[**RS.** ADDED REFERENCE TO SECTION 4.3 OF MY THESIS. THE PROPOSITON BELOW IS NOT PROVED IN MY THESIS. IT IS AN IMMEDIATE COROLLARY OF PROPOSITION 4.3.5 OF MY THESIS. I HAVE PROVED THIS RESULT IN THE PAPER THAT I AM WRITING ABOUT THE PROBABIISTIC MODEL.]]]

Proposition 2.3 *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 $\Theta \in \mathcal{F}_{H\lceil M_i}$, $P_{H\lceil M_i}[\Theta] = P_H[\{\alpha \in \Omega_H \mid \alpha\lceil M_i \in \Theta\}]$.* ■

2.4 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; it should not be possible to accomplish more by working less. In this section we present several results that apply to complexity functions; later in the paper we present results that apply only to complexity measures.

2.4.1 Expected Complexity

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

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

Complexity functions on full cuts enjoy several properties that are typical of random variables [4]. That is, if Θ is a full cut, then H induces a probability distribution $\mathcal{P}_\Theta$ over the states of Θ. In such case, ϕ is a random variable and $E_\phi[H,\Theta]$ is the expected value of the random variable.

2.4.2 Linear Combination of Complexity Functions

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 (cf. Proposition 2.4). We use this property for the time analysis of the protocol of Aspnes and Herlihy. That is, we express the time complexity of the protocol in terms of two other complexity measures (rounds and elementary coin flips), and then we use Proposition 2.4 to derive an upper bound on the expected time for termination based on upper bounds on the expected values of the other two complexity measures. The analysis of the other two complexity measures is simpler, and the relationship

between time and the other two complexity measures can be studied using known methods for ordinary nondeterministic systems, with no probability involved.

Proposition 2.4 *Let H be a probabilistic execution fragment of some probabilistic automaton M, and let Θ be a full cut of H. Let ϕ, ϕ_1, ϕ_2 be complexity functions, and c_1, c_2 be two constants such that, for each $\alpha \in \Theta$, $\phi(\alpha) \leq c_1\phi_1(\alpha)+c_2\phi_2(\alpha)$. Then $E_\phi[H, \Theta] \leq c_1 E_{\phi_1}[H, \Theta]+ c_2 E_{\phi_2}[H, \Theta]$.*

Proof. From the definition of $E_\phi[H, \Theta]$ and the relationship between ϕ, ϕ_1, and ϕ_2,

$$E_\phi[H, \Theta] \leq \sum_{q \in \Theta} (c_1\phi_1(q) + c_2\phi_2(q))P_H[C_q].$$

By a simple algebraic manipulation,

$$E_\phi[H, \Theta] \leq c_1 \sum_{q \in \Theta} \phi_1(q)P_H[C_q] + c_2 \sum_{q \in \Theta} \phi_2(q)P_H[C_q].$$

The two sums above coincide with the definitions of $E_{\phi_1}[H, \Theta]$ and $E_{\phi_2}[H, \Theta]$, respectively. Thus, $E_\phi[H, \Theta] \leq c_1 E_{\phi_1}[H, \Theta] + c_2 E_{\phi_2}[H, \Theta]$. ■

2.4.3 Computation Subdivided into Phases

In this section we study a property of complexity functions that becomes useful whenever a computation can be divided into phases. Specifically, suppose that in a system there are 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 difficult part of this result is that several phases may run concurrently.

The protocol of Aspnes and Herlihy works in *rounds*. At each round a special *coin flipping* protocol is run, and the coin flipper flips a number of elementary coins (*elementary coin flips*). The expected number of elementary coin flips is bounded by some known value c independent of the round number. We also know an upper bound k on the expected number of rounds that are started. If we view each round as a phase, then Proposition 2.5 below says that the expected number of elementary coin flips is upper bounded by ck.

Proposition 2.5 *Let M be a probabilistic automaton. Let $\phi_1, \phi_2, \phi_3, \ldots$ be a countable collection of complexity measures 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, $E_{\phi'}[H, \Theta] \leq cE_{\phi}[H, \Theta]$.

Proof. From the definition of ϕ',

$$E_{\phi'}[H, \Theta] = \sum_{q \in \Theta} \sum_{i>0} \phi_i(q) P_H[C_q]. \tag{1}$$

Since for each $q \in \Theta$ and each $i > \phi(q)$, $\phi_i(q) = 0$, Equation (1) can be rewritten as

$$E_{\phi'}[H, \Theta] = \sum_{q \in \Theta} \left(\phi_1(q) + \cdots + \phi_{\lfloor \phi(q) \rfloor}(q)\right) P_H[C_q], \tag{2}$$

which can be rearranged into

$$E_{\phi'}[H, \Theta] = \sum_{i>0} \left(\sum_{q \in \Theta | \phi(q) \geq i} \phi_i(q) P_H[C_q] \right). \tag{3}$$

For each $i > 0$, let η_i denote the set of minimal states q of H that are prefixes of some element of Θ and such that $\phi(q) \geq i$. Then, by breaking the inner summation of Equation (3),

$$E_{\phi'}[H, \Theta] = \sum_{i>0} \left(\sum_{q \in \eta_i} P_H[C_q] \left(\sum_{q' \in \Theta | q \leq q'} \phi_i(q') P_H[C_{q'}]/P_H[C_q] \right) \right). \tag{4}$$

Since for each $q \in \eta_i$, $\phi_i(q) = 0$ ($\eta_i \subseteq \Theta_i$) the innermost expression of the right hand side of Equation (4) is $E_{\phi_i}[H \triangleright q, (\Theta \cap C_q) \triangleright q]$. Since $H \triangleright q$ is a fair probabilistic execution fragment of M as well, $E_{\phi_i}[H \triangleright q, (\Theta \cap C_q) \triangleright q] \leq c$. Thus,

$$E_{\phi'}[H, \Theta] \leq \sum_{i>0} \left(\sum_{q \in \eta_i} c P_H[C_q] \right), \tag{5}$$

and since $\sum_{q \in \eta_i} P_H[C_q] = P_H[\eta_i]$,

$$E_{\phi'}[H, \Theta] \leq \sum_{i>0} P_H[\eta_i] c. \tag{6}$$

Observe that $P_H[\eta_i]$ is the probability that ϕ is at least i in Θ. Recall also that ϕ is a random variable for the probability space identified by Θ. Thus, by Proposition 2.1, part 2, $\sum_{i>0} P_H[\eta_i] \leq E_{\phi}[H, \Theta]$, and by substituting in (5), $E_{\phi'}[H, \Theta] \leq cE_{\phi}[H, \Theta]$. ■

2.4.4 Complexity Functions and Parallel Composition

[[[**RS.** CHANGED "IN A MODULAR WAY"]]] To verify properties in a modular way it is useful to derive complexity properties of complex systems based on complexity properties of the single components. Proposition 2.6 helps in doing this.

Proposition 2.6 *Let M be $M_1 \parallel M_2$, 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, $E_{\phi}[H,\Theta] \leq c$.*

Proof. Let H be a probabilistic execution fragment of M, and let H_i denote $H\lceil M_i$. Let Θ be a full cut of H. Build a discrete probability space $\mathcal{P}_i$ as follows: $\Omega_i = \{q\lceil M_i \mid q \in \Theta\}$, and for each $q' \in \Omega_i$, $P_i[q'] = P_H[\{q \in \Theta \mid q\lceil M_i = q'\}]$. We prove first that the probability space $\mathcal{P}_i$ is a *fringe* of H_i as defined in [16], where a fringe of H_i is a probability distribution $\mathcal{P}$ over the states of H_i such that, for each state q of H_i, $\sum_{q' \geq q} P[q] \leq P_{H_i}[C_q]$.

Consider a state q of H_i. Then, from the definition of $\mathcal{P}_i$, $\sum_{q' \geq q} P_i[q'] = \sum_{q' \in \Theta | q'\lceil M_i \geq q} P_H[C_{q'}]$. Since Θ is a cut of H, the right expression above is $P_H[\cup_{q' \in \Theta | q'\lceil M_i \geq q} C_{q'}]$. Furthermore, the event $\cup_{q' \in \Theta | q'\lceil M_i \geq q} C_{q'}$ is a subset of the inverse image under projection of C_q. Thus, by Proposition 2.3, $\sum_{q' \geq q} P_i[q'] \leq P_{H_i}[C_q]$. This completes the proof that $\mathcal{P}_i$ is a fringe.

Let $E_{\phi_i}[H_i, \mathcal{P}_i]$ denote $\sum_{q \in \Omega_i} \phi_i(q) P_i[q]$. Then, since for each finite execution fragment α of M, $\phi(\alpha) = \phi_i(\alpha\lceil M_i)$, we derive $E_{\phi}[H,\Theta] = E_{\phi_i}[H_i, \mathcal{P}_i]$. We need to show that $E_{\phi_i}[H_i, \mathcal{P}_i] \leq c$.

[[[**RS.** BY CONTRADICTION]]] Suppose for the sake of contradiction that $E_{\phi_i}[H_i, \mathcal{P}_i] > c$. Then there is a constant $k > 0$ such that $\sum_{q \in \Omega_i | length(q) \leq k} \phi_i(q) P_i[q] > c$. Consider the full cut Θ_k of H_i containing all the states q of H_i with length k and all the elements of Ω_{H_i} with length less than k. Then, by definition of Θ_k, $\sum_{q \in \Omega_i | length(q) \leq k} \phi_i(q) P_i[q] \leq E_{\phi_i}[H_i, \Theta_k]$. This means that $E_{\phi_i}[H_i, \Theta_k] > c$, contradicting the hypothesis that $E_{\phi_i}[H_i, \Theta_k] \leq c$. ■

The converse of Proposition 2.6 does not hold in general. In fact, even though for each probabilistic execution fragment H of M and each full cut Θ of H, $E_{\phi}[H,\Theta] \leq c$, there could be a probabilistic execution fragment H' of M_i and a full cut Θ' of H' such that $E_{\phi_i}[H',\Theta'] > c$. As an example, H' could be the projection of no probabilistic execution fragment of M. If $i = 1$, then H' could be a probabilistic execution fragment resulting from the interaction with an environment that M_2 does not provide.

2.5 Probabilistic Complexity Statements

A probabilistic complexity statement is a predicate that can be used to state whether all the fair probabilistic executions of a probabilistic automaton guarantee some reachability property within some time t 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.

Probabilistic complexity statements can also be decomposed into simpler statements, thus splitting the progress properties of a randomized system into progress properties that either are simpler to analyze or can be derived by analyzing a smaller subcomponent of the system.

Progress statements are introduced in [11, 15, 16]. In this section we specialize the theory of [16] to fair schedulers.

2.5.1 Probabilistic Complexity Statements

[[[**RS.** ADDED INFORMAL EXPLANATION.]]] A probabilistic complexity statement is a predicate of the form $U \xrightarrow[p]{\phi \leq c} U'$, where U and U' are sets of states, ϕ is a complexity function, and c is a nonnegative real number. Informally, the meaning of $U \xrightarrow[p]{\phi \leq c} U'$ is that starting from any state of U, under any fair scheduler, the probability of reaching a state from U' within complexity c is at least p. The complexity of an execution fragment is measured according to ϕ.

Definition 2.7 Let M be a probabilistic I/O automaton, $U, U' \subseteq \mathit{States}(M)$, $c \in \Re$, and ϕ be a complexity measure. Then $U \xrightarrow[p]{\phi \leq c} U'$ 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 $\mathit{lstate}(\alpha') \in U'$. ■

The fair probabilistic execution fragments of a probabilistic automaton enjoy a property that in [16] is called *finite history insensitivity*. Thus, using a result of [16], the following holds, which permits us to decompose a progress property into simpler progress properties.

Proposition 2.8 *Let M be a probabilistic automaton, and let $U, U', U'' \subseteq \mathit{States}(M)$. Let ϕ be a complexity measure. Then,*

1. *if $U \xrightarrow[p]{\phi \leq c} U'$ and $U' \xrightarrow[p']{\phi \leq c'} U''$, then $U \xrightarrow[pp']{\phi \leq c+c'} U''$;*
2. *if $U \xrightarrow[p]{\phi \leq c} U'$, then $U \cup U'' \xrightarrow[p]{\phi \leq c} U' \cup U''$.* ■

2.5.2 From Probabilistic Complexity Statements to Expected Complexity

In this section we show how to use probabilistic complexity statements to derive properties about expected complexities. In the analysis of the protocol of Aspnes and Herlihy we use

the result of this section to study the expected number of rounds that the protocol needs to terminate. The results of this section, as well as Proposition 2.8 in the previous section, rely on the subadditivity property of a complexity measure, that is, they are not applicable to general complexity functions. [[[**RS.** GENERAL]]]

Let M be a probabilistic automaton, and let $U, U' \subseteq States(M)$. We 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. That is, $\Theta_{U'}(H)$ represents the event that contains all those executions of Ω_H where a state from U' is reached. The following theorem, which is an instantiation of a more general result of [16], provides a way of computing the expected complexity for satisfying $\Theta_{U'}(H)$.

Theorem 2.9 ([16]) *Let M be a probabilistic automaton and ϕ be a complexity measure for M. Let r be a real number such that for each execution fragment of M of the form sas', $\phi(sas') \leq r$, that is, each transition of M can increase the complexity ϕ by at most r. 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 (i.e., the probability of δ in the transition enabled from q in H is 0). Furthermore, suppose that*

1. $U \overset{\phi \leq c}{\underset{p}{\Longrightarrow}} U'$ *and*
2. $U \Rightarrow U\, unless\, U'$.

Then, $E_\phi[H, \Theta_{U'}(H)] \leq (c + r)/p$.

Proof outline.

We omit the proof that $P_H[\Theta_{U'}(H)] = 1$. Consider the cut $\Theta = \Theta_{U'} \cup \Theta_{c+r}$, where $\Theta_{U'}$ is the subset of $\Theta_{U'}(H)$ of states q with $\phi(q) \leq c$, and Θ_{c+r} is the set of minimal states q of H such that $\phi(q) \geq c + r$ and such that no proper prefix of q is in $\Theta_{U'}(H)$ (cf. Figure 1). Since $P_H[\Theta_{U'}(H)] = 1$, Θ is a full cut. Then, from Item 1, $P_H[\Theta_{U'}] \geq p$. From Item 2, all the states of Θ_{c+r} are still elements of U, and thus the experiment above can be repeated from those points. Each experiment takes $c + r$ complexity units. Since we repeat a binary experiment until it succeeds, and since each time the probability of success is at least p, we expect to repeat the experiment $1/p$ times before being successful. Thus, the expected complexity for reaching U' is at most $(c + r)/p$.

It may be surprising to see that we start new experiments every $c + r$ complexity units rather than every c units. This is because $\Theta_{U'} \cup \Theta_c$ would not be a cut if H contains a

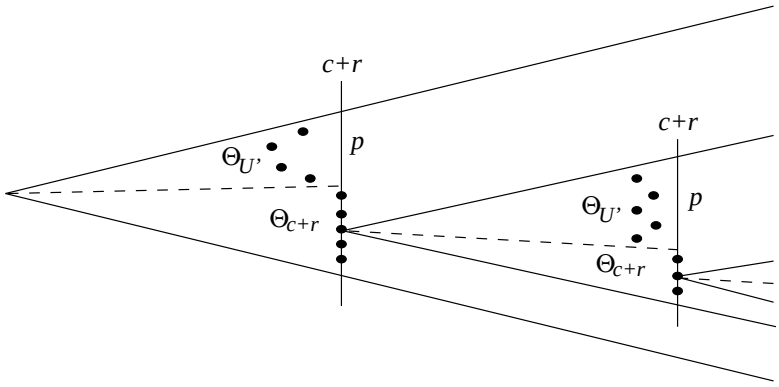

Figure 1: Computation of the expected time from U to U'.

transition that leaves from a c-complexity state and reaches a state from U' with probability p' and a $c+r$-complexity state with probability $1-p'$. For the fully detailed proof and for a more general result the reader is referred to [16]. ■

2.5.3 How to Verify Probabilistic Complexity Statements

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.

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.

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 carried out using the so-called *coin lemmas* [16], 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. The importance of coin lemmas is also in the fact that a common source of errors in the analysis

of a randomized algorithm is to map a probabilistic process onto a probabilistic execution in the wrong way, or, in other words, to believe that a probabilistic automaton always behaves like some defined probabilistic process while the claim is not true. In Section 3 we present a coin lemma that deals with random walks.

2.6 Refinement Mappings and Traces

A common verification technique consists of specifying a system as an I/O automaton or a probabilistic I/O automaton and then building an *implementation* of the specification. Typically the notion of implementation is identified by some form of language inclusion. The important fact is that the interesting properties of a specification are preserved by the notion of implementation, that is, whenever a property is true for the specification, such property is true for the implementation as well. In this section we provide the pieces of the technique that we use for the analysis of the algorithm of Aspnes and Herlihy. More details can be found in [12, 13, 16].

2.6.1 Traces and Trace Distributions

Trace and trace distributions are abstractions of the behavior of automata and probabilistic automata, respectively, that are based only on the sequences of external actions that the automata can provide. Several times, as is the case for the algorithm of Aspnes and Herlihy, the interesting properties of a system can be expressed in terms of trace and trace distributions. In such cases it is possible to use traces and trace distributions for the analysis and in particular to use the related proof techniques.

Let α be an execution of an automaton A. The *trace* of α, denoted by $trace(\alpha)$, is the ordered sequence of the external actions that appear in α. Denote a generic trace by β. A trace is *fair* if it is the trace of a fair execution. Denote by $traces(A)$ the set of traces of A and by $ftraces(A)$ the set of fair traces of A.

Let H be a probabilistic execution fragment of a probabilistic automaton M. Let $\Omega = ext(M)^* \cup ext(M)^\omega$ be the set of finite and infinite sequences of external actions of M. The *trace distribution* of H, denoted by $tdistr(H)$, is the probability space $(\Omega, \mathcal{F}, P)$ where $\mathcal{F}$ is the minimum σ-field that contains the set of cones C_β, where β is an element of $ext(M)^*$, and $P = trace(P_H)$, that is, for each $E \in \mathcal{F}$, $P[E] = P_H[\{\alpha \in \Omega_H \mid trace(\alpha) \in E\}]$. The fact that $tdistr(H)$ is well defined follows from standard measure theory arguments. In simple words, a trace distribution is just a probability distribution over traces induced by a probabilistic execution. Denote a generic trace distribution by $\mathcal{D}$. A trace distribution of a probabilistic automaton M is the trace distribution of one of the probabilistic executions of M. A trace distribution is *fair* if it is the trace distribution of a fair probabilistic execution. Denote by $tdistrs(M)$ the set of trace distributions of M and by $ftdistrs(M)$ the set of fair trace distributions of M.

2.6.2 Refinements

[[[**RS.** IMPLEMENTED CHANGES IN THE PARAGRAPH.]]] Denote a transition (s, a, s') by $s \xrightarrow{a}$. For a finite sequence $a_1 \cdots a_n$ let $s \xrightarrow{a_1 \cdots a_n} s'$ if there is a collection of states $s_1, \ldots, s_{n-1}$ such that $s \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \xrightarrow{a_{n-1}} s_{n-1} \xrightarrow{a_n} s'$. For any external action a, let $s \stackrel{a}{\Longrightarrow} s'$ if there are two finite sequences x, y of internal actions and two states s_1, s_2 such that $s \xrightarrow{x} s_1 \xrightarrow{a} s_2 \xrightarrow{y} s'$. Let $s \stackrel{\epsilon}{\Longrightarrow} s'$ if there is a finite sequence x of internal actions such that $s \xrightarrow{x} s'$.

Let A_1, A_2 be two automata with the same external actions. A *refinement* from A_1 to A_2 is a function $h : \mathit{States}(A_1) \to \mathit{States}(A_2)$ such that the following conditions hold.

1. For each $s \in \mathit{Start}(A_1)$, $h(s) \in \mathit{Start}(A_2)$.
2. For each transition $s \xrightarrow{a} s'$ of A_1, $h(s) \stackrel{a \lceil ext(A_2)}{\Longrightarrow} h(s')$.

That is, A_2 can simulate all the transitions of A_1 via the refinement function h. An important property of a refinement is the following.

Proposition 2.10 ([13]) *Suppose that there exists a refinement from A_1 to A_2. Then $\mathit{traces}(A_1) \subseteq \mathit{traces}(A_2)$.* ■

A refinement can be defined also for probabilistic automata as follows. Let M_1, M_2 be two probabilistic automata with the same external actions. A probabilistic *refinement* from M_1 to M_2 is a function $h : \mathit{States}(M_1) \to \mathit{States}(M_2)$ such that the following conditions hold.

1. For each $s \in \mathit{Start}(M_1)$, $h(s) \in \mathit{Start}(M_2)$.
2. For each $s \xrightarrow{a} \mathcal{P}$, $h(s) \stackrel{a \lceil ext(M_2)}{\Longrightarrow} h(\mathcal{P})$.

In particular, a refinement is a special case of a probabilistic refinement. The following property is valid as well.

Proposition 2.11 ([16]) *Suppose that there exists a probabilistic refinement from M_1 to M_2. Then $\mathit{tdistrs}(M_1) \subseteq \mathit{tdistrs}(M_2)$.* ■

Finally, the existence of refinements is preserved by parallel composition, thus enabling modular verification.

Proposition 2.12 ([16]) *Suppose that there exists a probabilistic refinement between two probabilistic automata M_1 and M_2. Then, for each probabilistic automaton M compatible with M_1 and M_2, there exists a probabilistic refinement from $M_1 \parallel M$ to $M_2 \parallel M$.* ■

2.6.3 The Execution Correspondence Theorem

Refinements can be used also to show some liveness properties. Specifically, it is possible to use refinements to derive fair trace inclusion and fair trace distribution inclusion. Our main technique is based on the *execution correspondence theorem* [6], which allows us to establish close relationships between the executions of two automata.

We use refinements in the analysis of the shared counter in the algorithm of Aspnes and Herlihy. Our analysis is carried out mainly on an abstract specification of the counters. This allows us to avoid dealing with unimportant details.

Let A_1 and A_2 be I/O automata with the same external actions and let h be a refinement from A_1 to A_2. For an execution fragment α, let $|\alpha|$ denote the number of actions that occur in α. If α is an infinite execution fragment, then $|\alpha|$ is ∞. Let $\alpha = s_0 a_1 s_1 a_2 s_2 \cdots$ and $\alpha' = u_0 b_1 u_1 b_2 u_2 \cdots$ be executions of A_1 and A_2, respectively. We say that α and α' are *h-related*, written $(\alpha, \alpha') \in h$, if there exists a total, nondecreasing mapping $m : \{0, 1, \ldots, |\alpha|\} \to \{0, 1, \ldots, |\alpha'|\}$ such that

1. $m(0) = 0$,
2. $h(s_i) = u_{m(i)}$ for all $0 \leq i \leq |\alpha|$,
3. $trace(b_{m(i-1)+1} \cdots b_{m(i)}) = trace(a_i)$ for all $0 < i \leq |\alpha|$, and
4. for all j, $0 \leq j \leq |\alpha'|$, there exists an i, $0 \leq i \leq |\alpha|$, such that $m(i) \geq j$.

Theorem 2.13 ([6]) *Let A_1 and A_2 be automata with the same external actions, and let h be a refinement from A_1 to A_2. Then, for each execution α_1 of A_1 there is an execution α_2 of A_2 such that $(\alpha_1, \alpha_2) \in h$.* ■

The execution correspondence theorem can be used to show fair trace inclusion as follows: given $(\alpha_1, \alpha_2) \in h$, show that α_2 is fair whenever α_1 is fair. In this case we also say that h *preserves the fair executions of* A_1.

The execution correspondence theorem can be extended to the probabilistic case as well [16]. We do not write the formal definitions in this paper; however, the following proposition can be proved easily from the results about execution correspondence of [16]. [[[**RS.** THE NOTION OF "PRESERVES THE FAIR EXECUTIONS" IS DEFINED ABOVE.]]]

Proposition 2.14 *Let A_1, A_2 be two I/O automata, and let M be a probabilistic I/O automaton compatible with A_1 and A_2. Let h be a refinement from A_1 to A_2 that preserves the fair executions of A_1. Then $ftdistrs(A_1 \parallel M) \subseteq ftdistrs(A_2 \parallel M)$.*

Proof outline. Since h is a refinement from A_1 to A_2, we can conclude from [16] that the following function is a probabilistic refinement from $A_1 \parallel M$ to $A_2 \parallel M$: $h'(s_{A_1}, s_M) = (h(s_{A_1}), s_M)$. That is, h' coincides with h on the states of A_1 and A_2 and is the identity function on the states of M. Let H_1 be a fair probabilistic execution of $A_1 \parallel M$. From the definition of h'-relation of [16], and from the definition of h', it is possible to build a fair probabilistic execution H_2 of $A_2 \parallel M$ such that $(H_1, H_2) \in h'$. Then, from [16], $tdistr(H_1) = tdistr(H_2)$. ■

3 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 known in the literature as random walk. The main problem is to make sure that the protocol indeed behaves as a random walk, or better, to make sure that the protocol has the same probabilistic properties as 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.

In this section we present a coin lemma for random walks. That is, we show that if we choose events within a probabilistic execution fragment according to some specific rules, then the chosen events are guaranteed to have properties similar to the properties of random walks. Then, by verifying that each one of the chosen events guarantees progress, a non-probabilistic property, we can derive probabilistic progress properties of the protocol.

We start by presenting the theory of random walks followed by a coin lemma for random walks. Then we present a result that relates expectations within a random walk to expectations within a probabilistic execution. This result is used in the analysis of the protocol of Aspnes and Herlihy to study the expected complexity of the coin flipping protocols. Finally, we instantiate our new coin lemma to the specific case that we need in the paper.

3.1 Random Walks

Let X be a probability space with sample set $\{-1, 1\}$ that assigns probability p to 1 and probability $q = (1 - p)$ to -1. Let $RW = (\Omega_{RW}, \mathcal{F}_{RW}, P_{RW})$ be the probability space built as follows. The sample set Ω_{RW} is the set $\{-1, 1\}^\omega$ of infinite sequences of numbers from $\{-1, 1\}$. For each finite sequence $x \in \{-1, 1\}^n$, let C_x, the *cylinder* with base x, be the set of elements from Ω_{RW} with common prefix x, and let $P_{RW}[C_x] = p^k q^{n-k}$, where k is the number of 1's in x. Then $\mathcal{F}_{RW}$ is the minimum σ-field that contains the set of cylinders, and P_{RW} is the unique extension to $\mathcal{F}_{RW}$ of the measure defined on the cylinders. The construction is justified by standard measure theory arguments. In other words, RW is a probability space on infinite sequences of independent experiments performed according to X.

Similarly to our probabilistic executions, define an event of $\mathcal{F}_{RW}$ to be *finitely satisfiable* if it is a union of cylinders. Furthermore, denote a finitely satisfiable event by a set Θ of incomparable finite sequences over $\{-1, 1\}$.

Consider a particle in the real line, initially at position z, and let X describe a move of the particle: -1 corresponds to decreasing by 1 the position of the particle, and 1 corresponds to increasing by 1 the position of the particle. An element of Ω_{RW} describes an infinite sequence of moves of the particle. The probability space RW describes a *random walk* of the particle.

An important random walk is a random walk with *absorbing barriers*, that is, a random walk that is considered to be successful or failed whenever the particle reaches some specified positions (absorbing barriers) of the real line. Consider two barriers B, T such that $B \leq z \leq T$. Then the following events are studied:

1. the particle reaches T before reaching B;
2. the particle reaches B before reaching T;
3. the particle reaches either absorbing barrier.

Formally, given a starting point z and a finite sequence $x = x_1 x_2 \cdots x_n \in \{-1, 1\}^n$ let $z_x = z + \sum_{i \leq n} x_i$ be the position of the particle after x. Then, the events 1, 2, and 3 above are finitely satisfiable and can be denoted by the following sets of finite sequences, respectively:

1. the set $\mathbf{Top}_{RW}[B, T, z]$ of minimal sequences $x \in \{-1, 1\}^*$ such that $z_x = T$ and for no prefix x' of x, $z_{x'} = B$;
2. the set $\mathbf{Bot}_{RW}[B, T, z]$ of minimal sequences $x \in \{-1, 1\}^*$ such that $z_x = B$ and for no prefix x' of x, $z_{x'} = T$;
3. the set $\mathbf{Either}_{RW}[B, T, z] = \mathbf{Top}_{RW}[B, T, z] \cup \mathbf{Bot}_{RW}[B, T, z]$.

The following results are known from random walk theory [4].

Theorem 3.1 *Let $p = q = 1/2$. Then*

1. $P[\mathbf{Top}_{RW}[B, T, z]] = (T - z)/(T - B)$;
2. $P[\mathbf{Bot}_{RW}[B, T, z]] = (z - B)/(T - B)$;
3. $P[\mathbf{Either}_{RW}[B, T, z]] = 1$. ■

For a finitely satisfiable event Θ that has probability 1 it is possible to study the average number of moves that are needed to satisfy Θ as follows:

$$E_{RW}[\Theta] = \sum_{x \in \Theta} length(x) P_{RW}[C_x].$$

From random walk theory [4] we know the following result.

Theorem 3.2 *Let $p = q = 1/2$. Then $E_{RW}[\mathbf{Either}_{RW}[B, T, z]] = -z^2 + (B + T)z - BT$.* ■

3.2 A Coin Lemma for Random Walks

We use a terminology that resembles coin flipping; thus, the number -1 is replaced by t (tail), the number 1 is replaced by h (head), p is replaced by p_h, and q is replaced by p_t. Let M be a probabilistic automaton and let $Acts = \{\mathit{flip}_1, \ldots, \mathit{flip}_n\}$ 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 $States(M)$. Suppose that for every transition $(s, \mathit{flip}_i, \mathcal{P})$ with an action flip_i the following hold:

$$\Omega \subseteq U_i^h \cup U_i^t, \tag{7}$$

$$P[U_i^h] = p_h \text{ and } P[U_i^t] = p_t. \tag{8}$$

The actions from $Acts$ represent coin flips, and the sets of states U_i^h and U_i^t represent the two possible outcomes of a coin flip labeled with flip_i. Since the sets $Acts$ and $\mathbf{S}$ are usually clear from the context, we omit them from our notation. We write $Acts$ and $\mathbf{S}$ explicitly only the first time each new notation is introduced.

3.2.1 The Coin Lemma

Let Θ be a finitely satisfiable event of RW, and let H be a probabilistic execution fragment of M. Given an execution α of H, let $x_{Acts,\mathbf{S}}(\alpha)$ be the ordered sequence of results of the coin flips that occur in α, e.g., if the i^{th} occurrence of an action from $Acts$ in α is an occurrence of flip_j that leads to a state from U_j^h, then the i^{th} element of $x(\alpha)$ is h, and if the i^{th} occurrence of an action from $Acts$ in α is an occurrence of flip_j that leads to a state from U_j^t, then the i^{th} element of $x(\alpha)$ is t. Observe that $x(\alpha)$ is finite if in α there are finitely many occurrences of actions from $Acts$.

Let $\mathcal{W}_{Acts,\mathbf{S}}(H, \Theta)$ be the set of executions α of Ω_H such that either $x(\alpha)$ has a prefix in Θ, or $x(\alpha)$ is a prefix of some element of Θ. [[[**RS.** We cannot say that x(alpha) is in theta rather than a prefix of some element in theta.]]] Informally, $\mathcal{W}(H, \Theta)$ contains all those executions of Ω_H where either the coin flips describe a random walk contained in the event denoted by Θ, or there is a way to fix the values of the unflipped coins so that a random walk of the event denoted by Θ is obtained. In other words, if we view the scheduler as a malicious adversary that tries to resolve the nondeterminism so that the probability of $\mathcal{W}(H, \Theta)$ is minimized, the scheduler does not gain anything by not scheduling coin flipping operations.

Lemma 3.3 *$\mathcal{W}(H, \Theta)$ is measurable in $\mathcal{P}_H$.*

Proof. The set $\mathcal{W}(H, \Theta)$ is the union of two sets: the set of executions α of Ω_H such that $x(\alpha)$ has a prefix in Θ, and the set of executions α of Ω_H such that $x(\alpha)$ is a prefix of some

element of Θ. The first set is a union of cones of the form C_α such that $x(\alpha) \in \Theta$; the second set is the complement of a union of cones, that is, C_α such that $x(\alpha)$ is not a prefix of any element of Θ. ■

We now prove that, no matter how the nondeterminism is resolved, the probability P_H of the event $\mathcal{W}(H, \Theta)$ is lower-bounded by the probability P_{RW} of the event Θ. That is, the probability of the mapping of the event Θ onto H is at least as large as the probability of Θ. We first prove our result for a special class of events Θ in Lemma 3.4. Then, we prove the full result in Theorem 3.5. [[[**RS.** MODIFIED END OF PARAGRAPH.]]]

Lemma 3.4 *Suppose that for each transition $(s, \mathit{flip}_i, \mathcal{P})$ of M, $P[U_i^h] = p_h$ and $P[U_i^t] = p_t$. If there is a limit k to the length of the elements of Θ, i.e., if there is a k such that for each element q of Θ, $|q| \leq k$, [[[**RS.** ADDED EXPLANATION]]] then $P_H[\mathcal{W}(H, \Theta)] \geq P_{RW}[\Theta]$.*

Proof. For notational convenience, for each state q of H let $\mathcal{P}_q^H$ denote the probability space associated with the unique transition that leaves from q in H.

We prove that $P_H[\overline{\mathcal{W}(H, \Theta)}] \leq 1 - P_{RW}[\Theta]$.

For each state q of H, each $i \in \{1, \ldots, n\}$, and each $j \in \{h, t\}$, denote by $\Omega(q, U_i^j)$ the set $\{(\mathit{flip}_i, q') \in \Omega_q^H \mid \mathit{lstate}(q') \in U_i^j\}$ of pairs where flip_i occurs and leads to a state of U_i^j, and for each action a let a denote also the set of pairs whose first element is a, that is, the event that action a occurs. For each $i \in \{1, \ldots, n\}$, let Θ_i be the set of states q of H such that no action flip_j, $1 \leq j \leq n$, occurs in q, and such that $P_q^H[\mathit{flip}_i] > 0$.

The proof is by induction on $\mathit{length}(\Theta)$, the maximum length of the elements of Θ. If $\mathit{length}(\Theta) = 0$, then either $\Theta = \emptyset$ or $\Theta = \{\epsilon\}$, where ϵ denotes the empty sequence. In the first case $\mathcal{W}(H, \Theta) = \emptyset$, and thus $P_H[\overline{\mathcal{W}(H, \Theta)}] = 1 - P_{RW}[\Theta] = 1$; in the second case $\mathcal{W}(H, \Theta) = \Omega_H$, and thus $P_H[\overline{\mathcal{W}(H, \Theta)}] = 1 - P_{RW}[\Theta] = 0$. For the inductive step, suppose that $\mathit{length}(\Theta) = k + 1$. Then,

$$P_H[\overline{\mathcal{W}(H, \Theta)}] = \sum_{i \in \{1, \ldots, n\}} \sum_{q \in \Theta_i} P_H[C_q] \left(\sum_{j \in \{h,t\}} \sum_{(\mathit{flip}_i, q') \in \Omega(q, U_i^j)} P_q^H[(\mathit{flip}_i, q')] P_{H \triangleright q'}[\overline{\mathcal{W}(H \triangleright q', \Theta \triangleright j)}] \right). \quad (9)$$

where $\Theta \triangleright j$ is the event Θ after performing j, that is, the set of the tails of the sequences of Θ whose head is j. Informally, to violate $\mathcal{W}(\Theta \triangleright j, H \triangleright q')$ with a non-empty Θ, it is necessary to flip at least once and then violate the rest of Θ. Observe that $\mathit{length}(\Theta \triangleright j) \leq k$. Thus, by induction, for each $j \in \{h, t\}$ and each state q' of H,

$$P_{H \triangleright q'}[\overline{\mathcal{W}(H \triangleright q', \Theta \triangleright j)}] \leq 1 - P_{RW}[\Theta \triangleright j]. \quad (10)$$

Using (10) in (9), and factoring $1 - P_{RW}[\Theta \triangleright j]$ out of the innermost summation, we obtain

$$P_H[\overline{\mathcal{W}(H,\Theta)}] \leq \sum_{i \in \{1,\ldots,n\}} \sum_{q \in \Theta_i} P_H[C_q] \left(\sum_{j \in \{h,t\}} P_q^H[\Omega(q, U_i^j)](1 - P_{RW}[\Theta \triangleright j]) \right). \tag{11}$$

Let $i \in \{1, \ldots, n\}$, and $j \in \{h, t\}$, and consider a state q of H. From the definition of the transition relation of a probabilistic execution fragment, there is a collection of transitions $(lstate(q), \mathit{flip}_i, \mathcal{P}_k)$ and a collection of probabilities p_{t_k} such that $\sum_k p_{t_k} = P_q^H[\mathit{flip}_i]$ and $P_q^H[\Omega(q, U_i^j)] = \sum_k p_{t_k} P_k[U_i^j]$. From hypothesis, for each k, $P_k[U_i^j] = p_j$. Thus, $P_q^H[\Omega(q, U_i^j)] = P_q^H[\mathit{flip}_i] p_j$. By substituting in (11),

$$P_H[\overline{\mathcal{W}(H,\Theta)}] \leq \sum_{i \in \{1,\ldots,n\}} \sum_{q \in \Theta_i} P_H[C_q] P_q^H[\mathit{flip}_i] \left(\sum_{j \in \{h,t\}} (1 - P_{RW}[\Theta \triangleright j]) p_j \right). \tag{12}$$

Observe that $\sum_{i \in \{1,\ldots,n\}} \sum_{q \in \Theta_i} P_H[C_q] P_q^H[\mathit{flip}_i]$ is the probability that some action flip_i occurs from in H, and hence its value is at most 1. Furthermore, observe that $\sum_{j \in \{h,t\}} p_j P_{RW}[\Theta \triangleright j] = P_{RW}[\Theta]$, that is, since $p_h + p_t = 1$, $\sum_{j \in \{h,t\}} p_j (1 - P_{RW}[\Theta \triangleright j]) = 1 - P_{RW}[\Theta]$. Thus, from (12),

$$P_H[\overline{\mathcal{W}(H,\Theta)}] \leq 1 - P_{RW}[\Theta]. \tag{13}$$

This completes the proof. ■

Theorem 3.5 *Suppose that for each transition $(s, \mathit{flip}_i, \mathcal{P})$ of M, $P[U_i^h] = p_h$ and $P[U_i^t] = p_t$. Then, $P_H[\mathcal{W}(H,\Theta)] \geq P_{RW}[\Theta]$.*

Proof. For each $k > 0$, let Θ_k be the set of elements of Θ whose length is at most k. Then, $\Theta = \cup_{k>0} \Theta_k$, and from the definition of $\mathcal{W}$, $\mathcal{W}(H,\Theta) = \cup_{k>0} \mathcal{W}(H,\Theta_k)$. Furthermore, for each $k > 0$, $\Theta_k \subseteq \Theta_{k+1}$, and $\mathcal{W}(H,\Theta_k) \subseteq \mathcal{W}(H,\Theta_{k+1})$. From simple arguments of measure theory, $P_{RW}[\Theta] = \lim_{k \to +\infty} P_{RW}[\Theta_k]$, and $P_H[\mathcal{W}(H,\Theta)] = \lim_{k \to +\infty} P_H[\mathcal{W}(H,\Theta_k)]$. From Lemma 3.4, for each $k > 0$, $P_H[\mathcal{W}(H,\Theta_k)] \geq P_{RW}[\Theta_k]$. Thus, $\lim_{k \to +\infty} P_H[\mathcal{W}(H,\Theta_k)] \geq \lim_{k \to +\infty} P_{RW}[\Theta_k]$, that is, $P_H[\mathcal{W}(H,\Theta)] \geq P[\Theta]$. ■

3.2.2 Expected Complexity of the Random Walk

The next theorem shows that the average length of a random walk is preserved by the mapping $\mathcal{W}$, that is, for fixed H and Θ, the expected number of coin flips that may occur in H without reaching Θ is bounded above by the expected number of coin flips necessary to reach Θ in RW. First we need a definition.

Definition 3.6 *Let Θ be an event in RW, and let M be a probabilistic automaton. For each finite execution fragment α of M, define $\phi(\alpha)$ to be the number of actions from Acts that occur in α if $x(\alpha)$ does not have any prefix in Θ, and to be the number of actions from Acts that occur in the minimum prefix α' of α such that $x(\alpha') \in \Theta$, otherwise.* ■

Informally, $\phi(\alpha)$ is the number of moves of the random walk that occur in α before satisfying the event denoted by Θ. In particular, if Θ is not satisfied yet within α, $\phi(\alpha)$ is the total number of moves of the random walk that occur in α. Observe that ϕ is a complexity function but not a complexity measure. [[[**RS.** FIXED PHI AND ADDED FINAL SENTENCE.]]]

Theorem 3.7 *Suppose that for each transition $(s, flip_i, \mathcal{P})$ of H, $P[U_i^h] = p$ and $P[U_i^t] = q$. Also, suppose that $P_{RW}[\Theta] = 1$. Let Θ' be a full cut of H. Then $E_\phi[H, \Theta'] \leq E_{RW}[\Theta]$.*

Proof. By definition, $E_\phi[H, \Theta'] = \sum_{q \in \Theta'} \phi(q) P_H[C_q]$.

From the definition of ϕ, if $q' \leq q$ and $x(q') \in \Theta$, then $\phi(q') = \phi(q)$. Thus, we can build a new full cut Θ'' obtained from Θ' by replacing each $q \in \Theta'$ such that $x(q)$ has a prefix in Θ with the minimum prefix q' of q such that $x(q') \in \Theta$ and obtain $E_\phi[H, \Theta'] = \sum_{q \in \Theta''} \phi(q) P_H[C_q]$. In particular, for no element q of Θ'' does the sequence $x(q)$ have a proper prefix in Θ.

Partition Θ'' into the set Θ''_p of states q such that $x(q)$ is a prefix of some element of Θ, and the set Θ''_n of states q such that $x(q)$ is not a prefix of any element of Θ. From the definition of Θ'', for no element q of Θ''_n $x(q)$ has a prefix in Θ. Thus, $\mathcal{W}(H, \Theta) \cap (\cup_{q \in \Theta''_n} C_q) = \emptyset$. Since from Theorem 3.1 $P_H[\mathcal{W}(H, \Theta)] = 1$, we derive that $P_H[\Theta''_n] = 0$, which means that Θ''_p is a full cut of H. Furthermore, since $\Theta''_p \subseteq \Theta''$, $E_\phi[H, \Theta'] \leq \sum_{q \in \Theta''_p} \phi(q) P_H[C_q]$, that is, $E_\phi[H, \Theta'] \leq E_\phi[H, \Theta''_p]$.

For each $k > 0$, let $\Theta_{<k}$ be the set of elements of Θ whose length is less than k, and let $\Theta_{\geq k}$ be the set of elements of Θ whose length is at least k. Similarly, let $\Theta''_{<k}$ be the set of elements q of Θ''_p such that $length(x(q)) < k$, and let $\Theta''_{\geq k}$ be the set of elements q of Θ''_p such that $length(x(q)) \geq k$.

Fix $k > 0$, and let $\alpha \in \mathcal{W}(H, \Theta_{<k}) \cap (\cup_{q \in \Theta''_p} C_q)$. Since $\alpha \in \mathcal{W}(H, \Theta_{<k})$, from the definition of ϕ for each finite prefix α' of α, $\phi(\alpha') < k$. From the definition of Θ''_p, $\alpha \in C_q$ for some $q \in \Theta''_p$ with $length(x(q)) < k$. Thus, $\mathcal{W}(H, \Theta_{<k}) \cap (\cup_{q \in \Theta''_p} C_q) \subseteq \cup_{q \in \Theta''_{<k}} C_q$, which implies $P_H[\mathcal{W}(H, \Theta_{<k}) \cap (\cup_{q \in \Theta''_p} C_q)] \leq P_H[\Theta''_{<k}]$. Since $P_H[\Theta''_p] = 1$, $P_H[\mathcal{W}(H, \Theta_{<k})] = P_H[\mathcal{W}(H, \Theta_{<k}) \cap (\cup_{q \in \Theta''_p} C_q)]$. This implies that $P_H[\mathcal{W}(H, \Theta_{<k})] \leq P_H[\Theta''_{<k}]$.

From Theorem 3.5, $P_H[\mathcal{W}(H, \Theta_{<k})] \geq P_{RW}[\Theta_{<k}]$, which, combined with the previous result, gives $P_H[\Theta''_{<k}] \geq P_{RW}[\Theta_{<k}]$. From this we derive that $E_\phi[H, \Theta''_p] = \sum_{i>0} P_H[\Theta''_{\geq k}] \leq \sum_{i>0} P_{RW}[\Theta_{\geq k}] = E_{RW}[\Theta]$, where the first and third steps follow from Proposition 2.1. Since, we have shown already that $E_\phi[H, \Theta'] \leq E_\phi[H, \Theta''_p]$, we conclude that $E_\phi[H, \Theta'] \leq E_{RW}[\Theta]$. ■

3.3 Instantiation of the Coin Lemma

In this section we instantiate the results of Section 3.2 with the events presented in Section 3.1. We also introduce a notation that is more suitable for the specific concepts that are described.

Given a finite execution fragment α of M, let $\mathit{Heads}_{\mathit{Acts},\mathbf{S}}(\alpha)$ denote the number of actions of the form flip_i in α whose post state is in the corresponding set U_i^h, and let $\mathit{Tails}_{\mathit{Acts},\mathbf{S}}(\alpha)$ denote the number of actions of the form flip_i in α whose post state is in the corresponding set U_i^t. Let $\mathit{Diff}_{\mathit{Acts},\mathbf{S}}(\alpha)$ denote $\mathit{Heads}_{\mathit{Acts},\mathbf{S}}(\alpha) - \mathit{Tails}_{\mathit{Acts},\mathbf{S}}(\alpha)$.

Definition 3.8 *For each probabilistic execution fragment H of M, let* $\mathbf{Top}[\mathit{Acts},\mathbf{S},B,T,z](H)$ *be the set of executions α of Ω_H such that either*

- $\exists_{\alpha' \leq \alpha}((z + \mathit{Diff}(\alpha') = T) \wedge \forall_{\alpha'' \leq \alpha'}(B < z + \mathit{Diff}(\alpha'')))$, *or*
- $\forall_{\alpha' \leq \alpha}(B < z + \mathit{Diff}(\alpha') < T)$ *and actions from Acts occur finitely many times in α.*

The event $\mathbf{Top}[\mathit{Acts},\mathbf{S},B,T,z](H)$ captures the situations where either $z + \mathit{Diff}(\alpha')$ reaches the top barrier T before the bottom barrier B, or the total number of "flips" is finite and $z + \mathit{Diff}(\alpha')$ reaches neither barrier.

Definition 3.9 *For each probabilistic execution fragment H of M, let* $\mathbf{Bot}[\mathit{Acts},\mathbf{S},B,T,z](H)$ *be the set of executions α of Ω_H such that either*

- $\exists_{\alpha' \leq \alpha}((z + \mathit{Diff}(\alpha') = B) \wedge \forall_{\alpha'' \leq \alpha'}(z + \mathit{Diff}(\alpha'') < T))$, *or*
- $\forall_{\alpha' \leq \alpha}(B < z + \mathit{Diff}(\alpha') < T)$ *and actions from Acts occur finitely many times in α.*

The event $\mathbf{Bot}[\mathit{Acts},\mathbf{S},B,T,z](H)$ captures the situations where either $z + \mathit{Diff}(\alpha')$ reaches the bottom barrier B before the top barrier T, or the total number of "flips" is finite and $z + \mathit{Diff}(\alpha')$ reaches neither barrier.

Definition 3.10 *For each probabilistic execution fragment H of M, let*
$\mathbf{Either}[\mathit{Acts},\mathbf{S},B,T,z](H) \triangleq \mathbf{Top}[\mathit{Acts},\mathbf{S},B,T,z](H) \cup \mathbf{Bot}[\mathit{Acts},\mathbf{S},B,T,z](H)$.

The event $\mathbf{Either}[\mathit{Acts},\mathbf{S},B,T,z](H)$ excludes those executions of M where infinitely many "flips" occur and $z + \mathit{Diff}(\alpha')$ reaches neither barrier.

Proposition 3.11 *Let H be a probabilistic execution fragment of M. 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. $P_H[\mathbf{Either}[B,T,z](H)] = 1$.

Proof.

1. From the definitions, the events $\mathbf{Top}[B,T,z](H)$ and $\mathcal{W}(H, \mathbf{Top}_{RW}[B,T,z])$ are the same. From Theorems 3.1 and 3.5, $P_H[\mathbf{Top}[B,T,z](H)] \geq (z-B)/(T-B)$.

2. From the definitions, the events $\mathbf{Bot}[B,T,z](H)$ and $\mathcal{W}(H, \mathbf{Bot}_{RW}[B,T,z])$ are the same. From Theorems 3.1 and 3.5, $P_H[\mathbf{Bot}[B,T,z](H)] \geq (T-z)/(T-B)$.

3. From the definitions, the events $\mathbf{Either}[B,T,z](H)$ and $\mathcal{W}(H, \mathbf{Either}_{RW}[B,T,z])$ are the same. From Theorems 3.1 and 3.5, $P_H[\mathbf{Either}[B,T,z](H)] = 1$. ■

We conclude with an instantiation of the result about expected complexities. Let ϕ_{Acts} be the complexity measure such that $\phi_{Acts}(\alpha)$ is the number of actions from *Acts* that occur in α. Define $\phi_{Acts,B,T,z}(\alpha)$ to be the truncation of ϕ_{Acts} at the point where one of the absorbing barriers is reached. That is, if there is no prefix α' of α such that $z + \mathit{Diff}(\alpha') \in \{B,T\}$, then $\phi_{Acts,B,T,z}(\alpha) = \phi_{Acts}(\alpha)$; otherwise, $\phi_{Acts,B,T,z}(\alpha) = \phi_{Acts}(\alpha')$, where α' is the minimum prefix of α such that $z + \mathit{Diff}(\alpha') \in \{B,T\}$. Observe that $\phi_{Acts,B,T,z}$ is not a complexity measure, but rather a complexity function.

Example 3.1 If $T = -B = 10$, $z = 0$, α_1 contains 5 flip actions, all giving tail, and α_2 contains 15 flip actions, all giving head, then $\phi_{Acts,B,T,z}(\alpha_1) = 5$, $\phi_{Acts,B,T,z}(\alpha_2) = 10$, while $\phi_{Acts,B,T,z}(\alpha_1 \frown \alpha_2) = 20$, which is greater than $10 + 5$. ■

Proposition 3.12 *Let H be a probabilistic execution fragment of M, and let Θ' be a full cut of H. Let z be chosen so that $B \leq z \leq T$. Then, $E_{\phi_{Acts,B,T,z}}[H,\Theta'] \leq -z^2 + (B+T)z - BT$.*

Proof. For each state q of H observe that $\phi_{Acts,B,T,z}(\alpha) = \phi(x(\alpha))$, where ϕ is the function defined in Definition 3.6 using the set Θ of minimal sequences of $\{-1,1\}^*$ such that either B or T is reached starting from z. From Theorem 3.7, $E_{\phi_{Acts,B,T,z}}[H,\Theta'] \leq E_{RW}[\Theta]$. From Theorem 3.2, $E_{RW}[\Theta] \leq -z^2 + (B+T)z - BT$, and therefore $E_{\phi_{Acts,B,T,z}}[H,\Theta'] \leq -z^2 + (B+T)z - BT$. ■

Part II: The Case Study

4 The Algorithm of Aspnes and Herlihy

4.1 The Consensus Problem

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

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

It is known from [5] that there is no deterministic algorithm for asynchronous processes that solves consensus and guarantees termination even in the presence of at most one single faulty process. However, the problem becomes solvable using randomization if we relax the termination condition and we replace it with the following condition.

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

The algorithm that we analyze in this paper is due to Aspnes and Herlihy [3] and relies on the theory of random walks. It terminates within expected polynomial time. We have chosen this algorithm because it is frequently cited in the literature and because it is among the most complicated randomized algorithms so far proposed. The complex structure of the algorithm allows us to show how modular verification techniques can be applied within a randomized framework.

4.2 Description of the Algorithm

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

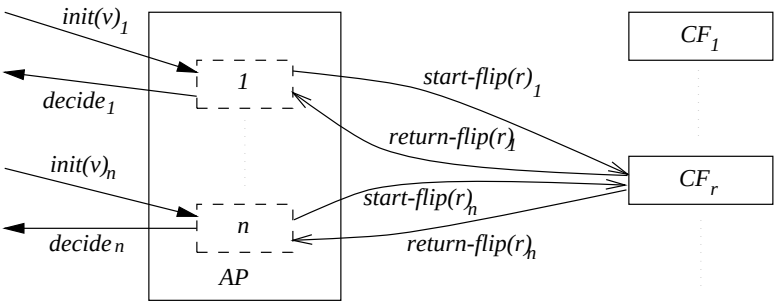

Figure 2: The interaction diagram of the algorithm of Aspnes and Herlihy.

at different rounds at some point of an execution. The variables $(value, round)$ are multiple-reader single-writer. Each process starts with its *round* field initialized to 0 and its *value* field initialized to $\perp$.

After receiving the initial value to agree on, each process i executes the following loop. It first reads the $(value, round)$ variables of all other processes in its local memory. We say that process i is a *leader* if according to its readings its own round is greater than or equal to the rounds of all other processes. We also say that a process i *observed* that another process j is a leader if according to i's readings the round of j is greater than or equal to the rounds of all other processes. If process i at round r discovers that it is a leader, and that according to its readings all processes that are at rounds r and $r-1$ have the same value as i, then i breaks out of the loop and decides on its value. Otherwise, if all processes that i observed to be leaders have the same value v , then i sets its value to v, increments its round and proceeds to the next iteration of the loop. In the remaining case (leaders that i observed do not agree), i sets its value to $\perp$ and scans the other processes again. If once again the leaders observed by i do not agree, then i determines its new preferred value for the next round by invoking a coin flipping protocol. There is a separate coin flipping protocol for each round. Figure 2 gives a high level view of the algorithm. The left box is the main algorithm which is subdivided into processes; the right boxes are the coin flipping protocols which interact with the main algorithm through some invocation and response messages.

[[[**RS.** *** NEED TO FIX PICTURE.]]] We represent the main part of the algorithm as an automaton AP (Agreement Protocol), [[[**RS.** NAME CONVENTIONS]]] and the coin flipping protocols as probabilistic automata CF_r (Coin Flipper), one for each round r. With this decomposition we can prove several important properties of the algorithm as properties of AP using ordinary techniques for non-probabilistic systems. Indeed, in this section we deal with AP only, and we leave the coin flippers unspecified. Table 1 describes the state variables of AP. The shared state of process i consists of a single-writer multiple-reader shared variable with two fields, $value(i)$ and $round(i)$, that contain process i's current preferred value and round. The local state of a process i consists of a program counter pc, two arrays, *values* and *rounds* that store the $(value, round)$ variables of other processes after i reads them, a variable *obs* that records the processes already observed by i, a variable *start* that records the initial preferred

Name	Values	Initially
Local state		
pc	$\{nil, init, read1, read2, check1, check2, flip, wait, decide\}$	*init*
values	$array[1 \ldots n]$ of $\{0, 1, \perp\}$	array of $\perp$
rounds	$array[1 \ldots n]$ of *int*	array of 0
obs	set of $\{1, \ldots, n\}$	$\emptyset$
start	$\{0, 1, \perp\}$	$\perp$
decided	*Bool*	*false*
stopped	*Bool*	*false*
Single-writer multiple-reader shared variables		
$(value(i), round(i))$	$\{0, 1, \perp\} \times int$	$(\perp, 0)$

Table 1: The state variables of a process i in AP.

value of i, and two boolean flags, *decided* and *stopped*, that reflect whether i has decided or failed. [[[**RS.** Explained role of stopped.]]] The variable *stopped* is not relevant for the actual code for process i; it is used only in the analysis of the algorithm to identify those points where process i has failed.

Table 2 describes the actions and the transition relation of AP. The transitions associated with each action a are described by giving the conditions that a state s should satisfy to enable a (Pre:), and the transformations that are performed on s to obtain the post-state of the transition (Eff:). If the precondition is omitted, then it is taken to be true. Table 2 is based on the following predicates and functions: *obs-max-round* is the maximum round observed by process i; $obs\text{-}leader(j)$ is true if i observes that j is a leader; $obs\text{-}agree(r, v)$ is true if the observations of all the processes whose round is at least r agree on v; $obs\text{-}leader\text{-}agree(v)$ is true if, according to the observations of i, the leaders agree on v; *obs-leader-value* is the value of one of the leaders observed by i. Formally,

$$
\begin{array}{rcl}
obs\text{-}max\text{-}round & \triangleq & \max_{j \in obs}(rounds[j]) \\
obs\text{-}leader(j) & \triangleq & j \in obs \wedge rounds[j] = obs\text{-}max\text{-}round \\
obs\text{-}agree(r, v) & \triangleq & \forall_{j \in obs}\, rounds[j] \geq r \Rightarrow values[j] = v \\
obs\text{-}leader\text{-}agree(v) & \triangleq & obs\text{-}agree(obs\text{-}max\text{-}round, v) \\
obs\text{-}leader\text{-}value & \triangleq & \begin{cases} v & \text{if } obs\text{-}leader\text{-}agree(v) \\ \text{undefined} & \text{if } \nexists_v\, obs\text{-}leader\text{-}agree(v) \end{cases}
\end{array}
$$

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 $obs\text{-}leader(i) \wedge$
      $\exists_{v \in \{0,1\}} obs\text{-}agree(rounds[i] - 1, v)$ 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 $pc = wait$ and $round(i) = r$ 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 2: The actions and transition relation of AP.

It is simple to check that *obs-leader-value* is a well defined function since it is never the case that *obs-leader-agree*(0) and *obs-leader-agree*(1) are satisfied simultaneously.

We associate all the locally controlled actions of a process i with a single task. Thus, an execution fragment α of AP is fair if all processes that are continuously enabled are scheduled eventually in α. [[[**RS.** REMOVED DEFINITION OF 1-LIVE.]]]

5 Proving Validity

[[[**RS.** TOUCHED PARAGRAPH.]]] The proof of validity is very simple and is based on an invariant property (cf. Invariant 5.2). In this section and in the rest of this paper we use the word "invariant" both for automata and for execution fragments. An invariant of an automaton is a property that is valid in all the reachable states of the automaton; an invariant of an execution fragment is a property that is valid in all the states of the execution fragment. For notational convenience, given $v \in \{0,1\}$, we denote by $\overline{v}$ the value $(v+1)\ mod\ 2$. We also define a new predicate:

$$agree(r,v) \quad \triangleq \quad \forall_j(round(j) \geq r \Rightarrow value(j) = v).$$

That is, predicate $agree(r,v)$ is true if all the processes at round at least r agree on value v.

Invariant 5.1 *Let α be an execution of AP where no action of the form $init(\bar{v})_i$ occurs. Then each state of α satisfies $agree(1,v)$ and obs-agree$(1,v)$.*

Proof. Straightforward inductive argument. Informally, each process observes that the leaders agree on v, and thus no process ever flips a coin or chooses $\bar{v}$ as its preferred value for the next round. ■

Invariant 5.2 *For each reachable state of AP, and each pair of processes i,j,*

1. *$s.round(i) = 0 \Rightarrow s.value(i) = \perp$, and*
2. *$s.rounds[i]_j = 0 \Rightarrow s.values[i]_j = \perp$.*

Proof. Straightforward inductive argument. ■

Theorem 5.3 (Validity property) *Let α be an execution of AP where no action of the form $init(\bar{v})_i$ occurs. Then in α no action of the form $decide(\bar{v})_i$ occurs.*

Proof. Suppose by contradiction that there is an occurrence of action $decide(\bar{v})_i$ in α, and let s be the state immediately before action $decide(\bar{v})_i$. From the transition relation of AP, $s.values[i]_i = \bar{v}$, and by Invariant 5.2, $s.rounds[i]_i > 0$. This contradicts Invariant 5.1. ■

6 Proving Agreement

In this section we prove the agreement property of AP, that is, that any two processes that decide within an execution decide the same value (cf. Theorem 6.2). We give the high level proof in Section 6.1 and we prove the main invariant in Section 6.2.

6.1 High Level Proof

The key idea of the agreement proof 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. We formalize this statement in Invariant 6.1.

Invariant 6.1 *Let i be a process. 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 6.1 states that if process i has observed all the other processes and has determined that it is a leader and that all the processes at round at least $r-1$ agree on a value v, then all the processes at round at least r agree on a value v. Before giving the proof of Invariant 6.1, we use Invariant 6.1 to prove the agreement property. Essentially the idea is that the premise of Invariant 6.1 is stable, that is, it is always satisfied in the future once it is satisfied: if process i satisfies the premise of Invariant 6.1, then process i decides on value v, and thus the local state of process i does not change any more. [[[**RS.** TOUCHED PARAGRAPH.]]]

Theorem 6.2 (Agreement property) *For every trace γ of AP the following is true: if $decide(v)_i$ and $decide(v')_j$ both occur in γ then $v = v'$.*

Proof. Let γ be a trace of AP such that $decide(v)_i$ and $decide(v')_j$ both occur in γ. Let α be an execution of AP that has trace γ. Assume without loss of generality that $decide(v)_i$ occurs first in γ. Let s_i and s_j be the states before actions $decide(v)_i$ and $decide(v')_j$ occur, respectively. From the transition relation of AP, process i satisfies the premise of Invariant 6.1 in state s_i, and process j satisfies the premise of Invariant 6.1 in state s_j. Thus, $s_i.agree(round(i), v)$ and $s_j.agree(round(j), v')$. Furthermore, it is a simple inductive argument to show that the premise of Invariant 6.1 is stable, that is, once it is satisfied it continues to be satisfied. Thus, $s_j.agree(round(i), v)$. Since in s_j there is at least one process at round $max(s_j.round(i), s_j.round(j))$, we derive that $v = v'$. ■

6.2 Proof of Invariant 6.1

The problem with Invariant 6.1 is that it is not strong enough to hold inductively. Therefore, we provide a stronger invariant (cf. Invariant 6.3) that implies Invariant 6.1 and holds inductively. Invariant 6.1 guarantees that some properties hold [[[**RS.** REMOVED ONLY]]] for those states where a process i has observed all other processes; for the inductive argument we need to guarantee some properties also for those states where process i has not observed all other processes yet. Furthermore, we need to ensure more properties than just the fact that all processes at round at least r have value v. In particular, we need to make sure that all processes at round $r-1$ cannot reach round r with a value different from v.

[[[**RS.** THE DEFINITION OF V BAR IS TOO SPECIFIC. I WOULD NOT PUT IT IN A PRELIMINARY SECTION. I HAVE IMPLEMENTED THE OTHER CHANGES SUGGESTED FOR THIS PARAGRAPH.]]] Given $v \in \{0,1\}$, denote by $\overline{v}$ the value $(v+1)$ *mod* 2. Define new predicates and functions *fill-max-round*$_i$, *fill-leader*$(j)_i$, *fill-agree*$(r,v)_i$, and *fill-leader-agree*$(v)_i$ to be the same as the corresponding predicates and functions *obs-max-round*$_i$, *obs-leader*$(j)_i$, *obs-agree*$(r,v)_i$, and *obs-leader-agree*$(v)_i$, with the following exception: the rounds and preferred values used in the definitions are the values observed by i for the processes that i has already observed, and the actual values of the shared variables for the processes that i has not yet observed. In other words, an incomplete observation is "completed instantly" with the actual values of the unobserved processes. Formally, for each process i, let *fill-rounds*$_i$ and *fill-values*$_i$ be two vectors defined as follows: [[[**NL.** CONVENTION ABOUT PUNCTATION IN DISPLAYED FORMULAS?]]] [[[**RS.** NONE BEFORE. NOW I PUT A COMMA AT THE END OF EACH LINE EXCEPT FOR THE LAST LINE.]]]

$$\textit{fill-rounds}[j]_i \triangleq \begin{cases} \textit{rounds}[j]_i & \text{if } j \in \textit{obs}_i, \\ \textit{round}(j) & \text{if } j \notin \textit{obs}_i, \end{cases}$$

$$\textit{fill-values}[j]_i \triangleq \begin{cases} \textit{values}[j]_i & \text{if } j \in \textit{obs}_i, \\ \textit{value}(j) & \text{if } j \notin \textit{obs}_i. \end{cases}$$

The vectors *fill-rounds* and *fill-values* are called the *filled* vectors of rounds and values. Then,

$$\begin{aligned} \textit{fill-max-round}_i &\triangleq \max_j(\textit{fill-rounds}[j]_i), \\ \textit{fill-leader}(j)_i &\triangleq \textit{fill-rounds}[j]_i = \textit{fill-max-round}_i, \\ \textit{fill-agree}(r,v)_i &\triangleq \textit{fill-rounds}_i[j] \geq r \Rightarrow \textit{fill-values}[j]_i = v, \\ \textit{fill-leader-agree}(v)_i &\triangleq \textit{fill-agree}(\textit{fill-max-round}_i, v)_i. \end{aligned}$$

The actual invariant that we prove is the following.

Invariant 6.3 *Let i be a process. Given a reachable state of AP, let $v = value(i)$, $r = round(i)$. If the following holds*

1. *obs-agree$(r-1, v)_i$,*
2. *fill-agree$(r, v)_i$,*
3. *fill-max-round$_i = r$,*

then

a. $\forall_j$ *obs-agree$(r, v)_j$,*

b. *agree(r, v),*

c. $\forall_{j \in obs_i}((round(j) = r-1 \wedge value(j) \neq v) \Rightarrow$ *fill-max-round$_j \geq r$).*

[[[**RS.** Added explanation of preconditions. Changed also following paragraph.]]] Informally, Invariant 6.3 states that if nothing is preventing some process i from deciding on a value v at round r, then none of the processes observed by i is in a position to cause other processes not to agree on v at round r. Thus, the premises state that according to the observations of process i, process i is a leader at round r and observes that the other processes that are at round at least $r-1$ agree on v; furthermore all the non-observed processes do not compromise the leadership of process i and agree on v if they are at round at least r. This means that it is possible for i to decide on v after completing its scan: the non-observed processes that are at round $r-1$ and do not agree on v may reach round r with value v before being observed by i. Condition a states that all processes observe agreement on v from round r, Condition b states that all processes at round at least r do agree on v, and Condition c states that none of the processes that have been observed already by process i is in a condition to reach round r with a value different from v.

At this point we can understand better the use of $\perp$ in AP. When a process i is about to decide on v at round r, it could be the case that another process j at round $r-1$ is about to flip a coin for the value to be used in round r. Process j could have observed some old values of the other processes. However, in such a case the value of process j would be $\perp$, and Condition c, together with Condition a, ensures that process j observes that the leaders agree on v. Thus, process j cannot flip. In other words, a process j may not discover that another process i is about to decide on v at round r during its first scan; however, process j would certainly discover the intent of process i during its second scan.

Observe that Invariant 6.3 implies Invariant 6.1 directly; thus, proving Invariant 6.3 is sufficient to prove Invariant 6.1. To prove Invariant 6.3 we need several auxiliary invariants that illustrate some of the key ideas behind the algorithm. Several invariants have straightforward inductive proofs, which we omit. The first invariant, Invariant 6.4, states that a process that has not started yet is at round 0.

Invariant 6.4 *Let i be a process. Then, for each reachable state of AP,*

$$(pc_i = init) \Rightarrow (round(i) = 0).$$ ■

Invariant 6.5 states that a process has observed all other processes whenever either it has decided, or it is checking the local variables, or it is interacting with the coin flipping protocol.

Invariant 6.5 *Let i be a process. Then, for each reachable state of AP,*

$$pc_i \in \{check1, check2, decide, flip, wait\} \Rightarrow obs_i = \{1, \ldots, n\}.$$ ■

Invariant 6.6 states that the preferred value of a process is $\perp$ during the second scan of the shared variables and during the interaction with the coin flipping protocol.

Invariant 6.6 *Let i be a process. Then, for each reachable state of AP,*

$$pc_i \in \{read2, check2, flip, wait\} \Rightarrow value(i) = \perp.$$ ■

Invariant 6.7 states that if a process is interacting with a coin flipping protocol, then that process observes that the leaders do not agree.

Invariant 6.7 *Let i be a process. Then, for each reachable state of AP,*

$$pc_i \in \{flip, wait\} \Rightarrow \nexists_v obs\text{-}leader\text{-}agree(v)_i.$$ ■

Invariant 6.8 states that the round numbers observed by each process are never larger than the actual round numbers of the processes.

Invariant 6.8 *Let i, j be two processes. Then, for each reachable state of AP,*

$$rounds[j]_i \leq round(j).$$ ■

Invariant 6.9 is a consequence of the fact that a process cannot prefer two different values during the same round. That is, if process j observes the current round of process i and process i does not prefer $\perp$, then then the value of process i observed by process j coincides with the actual preferred value of process i. In other words, if process j observes that at some point process i is at round r and prefers value v, then the actual preferred value of process i while its round is r is either v or $\perp$.

Invariant 6.9 *Let i, j be two processes. Then, for each reachable state of AP,*

$$(rounds[i]_j = round(i) \wedge value(i) \in \{0, 1\}) \Rightarrow (values[i]_j = value(i)).$$

Proof. For notational convenience, let $I(s)$ denote the invariant above. We prove $I(s)$ by induction on the length of an execution of AP leading to s. If s is a start state, then $I(s)$ is satisfied trivially since $s.value(i) = \perp$ for all i. For the inductive step it is enough to show that for every transition (s, a, s') of AP, $I(s)$ implies $I(s')$. We distinguish the following cases based on a.

1. $a = read1(i)_j$ or $a = read2(i)_j$.

   The transition relation of AP ensures that $s'.values[i]_j = s'.value(i)$. Thus, $I(s')$ is true.

2. $a = check1_i$ or $a = check2_i$ or $a = start(v)_i$, or $a = return\text{-}flip(v, r)_i$, $v \in \{0, 1\}$, $r > 0$.

   If $s'.pc_i = decide$, then none of the relevant variables for $I(s')$ has changed, and thus $I(s')$ is true; if $s'.pc_i \neq decide$, then either $s'.round(i) = s.round(i) + 1$ or $s'.value(i) = \perp$ (cf. Ivariants 6.4 and 6.6). In the first case, since process j does not change state, and since by Invariant 6.8 $s.round(i) \geq s.rounds[i]_j$, we derive that $s'.round(i) > s'.rounds[i]_j$. Thus, in both cases one of the premises of $I(s')$ is not satisfied, which means that $I(s')$ is true.

3. None of the previous cases hold.

   $I(s)$ implies $I(s')$ trivially, since all the relevant components stay unchanged. ■

Invariant 6.10 states that whenever a process has observed itself, the observed round and value coincide with the actual round and value. [[[**RS.** TOUCHED EXPLANATION]]]

Invariant 6.10 *Let i be a process. Then, for each reachable state of AP,*

$$i \in obs_i \Rightarrow (rounds[i]_i = round(i) \wedge values[i]_i = value(i)).$$

Proof. Fix a process i. For notational convenience let $I(s)$ denote the invariant above. We prove $I(s)$ by induction on the length of an execution of AP leading to s. If s is a start state, then $I(s)$ is satisfied trivially since $s.obs_i = \emptyset$. For the inductive step it is enough to show that for every transition (s, a, s') of AP, $I(s)$ implies $I(s')$. We distinguish the following cases based on a.

1. $a = read1(i)_i$ or $a = read2(i)_i$.

   The transition for $read(i)_i$ ensures that $s'.rounds[i]_i = s.round(i)$ and that $s'.values[i]_i = s.value(i)$. Since $round(i)$ and $value(i)$ do not change from s to s', $I(s')$ is true.

2. $a = check1_i$ or $a = check2_i$ or $a = init(v)_i$, or $a = return\text{-}flip(v, r)_i$, $v \in \{0, 1\}$, $r > 0$.

   If $s'.pc_i \in \{decide, flip\}$, then none of the relevant variables for $I(s')$ has changed from s to s', and $I(s')$ is true. If $s'.pc_i \notin \{decide, flip\}$, then $s'.obs = \emptyset$, falsifying $i \in s.obs_i$. Therefore, $I(s')$ is satisfied trivially.

3. None of the cases above hold.

   $I(s)$ implies $I(s')$ trivially, since all the relevant conditions stay unchanged. ■

Invariant 6.11 states that whenever the maximum round is at most r and all processes agree on a value v from round r, then all processes observe that there is agreement on v from round r. [[[**RS.** TOUCHED EXPLANATION]]]

Invariant 6.11 *Let r be a non-negative integer and $v \in \{0,1\}$. Then, for each reachable state of AP,*

$$(\textit{max-round} \leq r \wedge \textit{agree}(r,v)) \Rightarrow \forall_j \textit{obs-agree}(r,v)_j.$$

Proof. Suppose that the premises of the invariant above are satisfied, and let i, j be two processes such that $\textit{rounds}[i]_j = r$. By Invariant 6.8 and from $\textit{max-round} \leq r$, $\textit{round}(i) = r$. Thus, from $\textit{agree}(r,v)$, $\textit{value}(i) = v$. By Invariant 6.9, $\textit{values}[i]_j = v$. ■

The following lemma is more technical and is used to shorten the inductive argument in the proof of Invariant 6.3. It states that, under certain conditions, if the premises of Invariant 6.3 are satisfied in the post-state of a transition, then the premises of Invariant 6.3 are satisfied in the pre-state of the transition as well.

Lemma 6.12 *Let (s, a, s') be a transition of AP, where a is either $\textit{read}1(k)_j$ or $\textit{read}2(k)_j$ or $\textit{check}1_j$ or $\textit{check}2_j$ or $\textit{return-flip}(v', r')_i$, $v' \in \{0,1\}$, $r' > 0$. Let i be a process such that $i \neq j$ if $a = \textit{check}1_j$ or $a = \textit{check}2_j$ or $a = \textit{return-flip}(v', r')_j$. If, for $v \in \{0,1\}$ and $r > 0$, the following conditions hold in s':*

1. $\textit{obs-agree}(r-1, v)_i$,
2. $\textit{fill-agree}(r, v)_i$,
3. $\textit{fill-max-round}_i = r$,
4. $\textit{value}(i) = v$ *and* $\textit{round}(i) = r$,

then the same conditions hold in s as well.

Proof. We distinguish two cases based on a.

1. $a = \textit{read}1(k)_j$ or $a = \textit{read}2(k)_j$.

   Observe that for each process l, $s.\textit{value}(l) = s'.\textit{value}(l)$ and $s.\textit{round}(l) = s'.\textit{round}(l)$. This implies Condition 4 in s. It is left to show Conditions $1, 2$, and 3 for s. If $i \neq j$ then $s.\textit{values}_i = s'.\textit{values}_i$ and $s.\textit{rounds}_i = s'.\textit{rounds}_i$. Thus, Conditions $1, 2$,

and 3 are satisfied trivially in s. If $i = j$, then for every process l such that $l \neq k$, $s.values[l]_i = s'.values[l]_i$ and $s.rounds[l]_i = s'.rounds[l]_i$. Since $k \notin s.obs_i$ (i is reading from k), and since Condition 1 holds in s', Condition 1 also holds in s. Condition 2 follows directly from Condition 2 for s' and the fact that $s'.values[k]_i = s.value(k)$ and $s'.rounds[k]_i = s.round(k)$; Condition 3 follows from Condition 3 in s' and from $s'.rounds[k]_i = s.round(k)$.

2. $a = check1_j$ or $a = check2_j$ or $a = return\text{-}flip(v', r')_j$.

   Observe that, by Invariants 6.5 and 6.10, $s.round(j) = s.rounds[j]_j$. Conditions 3 and 4 are trivial, since the state of process i is the same in s and s' ($i \neq j$), $s.round(j) \leq s'.round(j)$, and $s.round(i) = r$. Similarly, Condition 1 holds in s. It is left to show that Condition 2 holds in s. Since j is the only process that changes state, and since Condition 2 is affected only if $j \notin s'.obs_i$, which is equivalent to $j \notin s.obs_i$, it is sufficient to verify $s.round(j) = r \Rightarrow s.value(j) = v$ under the assumption that $j \notin s.obs_i$. We distinguish two cases.

   (a) $s'.pc_j \in \{decide, flip\}$.
   No other state variable has changed in the transition. Thus, Condition 2 holds in s.

   (b) $s'.pc_j = read$.
   From Condition 3 in s' we have $s'.round(j) \leq r$. If $s'.value(j) = \perp$, then Condition 2 for s' implies $s'.round(j) < r$, and therefore $s.round(j) < r$, which implies Condition 2 for s. If $s'.value(j) \neq \perp$, then the transition relation of AP implies $s.round(j) < s'.round(j)$, and therefore, since from Condition 3 $s'.round(j) \leq r$, $s.round(j) < r$. This implies Condition 2 for s. ■

Proof of Invariant 6.3
For notational convenience, for each state s and process i let $I(s)$ denote the whole invariant, $C1(s,i)$, $C2(s,i)$, and $C3(s,i)$ denote Conditions 1, 2, and 3, respectively, and $Ca(s,i)$, $Cb(s,i)$, and $Cc(s,i)$ denote Conditions a, b, and c, respectively.

We prove $I(s)$ by induction on the length of an execution of AP leading to s. If s is a start state, then $I(s)$ is satisfied trivially since $s.value(j) = \perp$ for all j and $s.obs_i = \emptyset$, and thus $C2(s,i)$ is not satisfied. For the inductive step it is enough to show that for every transition (s, a, s') of AP, $I(s)$ implies $I(s')$. We distinguish the following cases based on a.

1. $a = start(v')_j$ for some v' and j.

   Consider a processes i such that $C1(s',i) \wedge C2(s',i) \wedge C3(s',i)$. Let $r = s'.round(i)$, $v = s'.round(i)$. We distinguish the following cases.

   (a) $i = j$.
   In this case $r = 1$ and $v' = v$. Since $s'.obs_i = \emptyset$, $Cc(s',i)$ is trivially true, and $Cb(s',i)$ follows from $C2(s',i)$. Furthermore, from $C3(s',i)$, $s'.max\text{-}round = 1$, and thus the premises of Invariant 6.11 are satisfied, giving $Ca(s',i)$.

(b) $i \neq j$ and $r = 1$.

From $C1(s', i)$, $j \notin s'.obs_i$, otherwise process i would have observed $\perp$ at round $r-1$. Thus, from $C2(s', i)$, $v' = v$. Since, except for process j, all the relevant components for $C1(s, i)$ and $C2(s, i)$ do not change, we derive $C1(s, i) \wedge C2(s, i)$. If $C3(s, i)$ is true as well, then $Ca(s, i) \wedge Cb(s, i) \wedge Cc(s, i)$ is true, and $Ca(s', i) \wedge Cb(s', i) \wedge Cc(s', i)$ follow directly. If $C3(s, i)$ is false, then $s.obs_i = \emptyset$, otherwise $C1(s, i)$ would be false, and thus j is the only process in s' that is at round r. This implies $Cb(s', i) \wedge Cc(s', i)$ directly. By Invariant 6.8, $Ca(s, i)$ is true, and thus, since none of the relevant state components change, $Ca(s', i)$ is true as well.

(c) $i \neq j$ and $r = 2$.

Observe that $C1(s, i) \wedge C2(s, i) \wedge C3(s, i)$ is true, since process j does not affect their validity. Thus, $Ca(s, i) \wedge Cb(s, i) \wedge Cc(s, i)$ is true. Then, $Ca(s', i) \wedge Cb(s', i)$ since process j does not affect their validity. Since $s'.obs_j = \emptyset$, from $C3(s', i)$ and by Invariant 6.8 we derive that process j satisfies the condition for $Cc(s', i)$. Thus, $Cc(s', i)$ follows from $Cc(s, i)$.

(d) $i \neq j$ and $r > 2$.

$I(s')$ follows trivially from $I(s)$ since process j does not affect any of the relevant conditions.

2. $a = read1(k)_j$ or $a = read2(k)_j$ for some j and k.

Consider a processes i such that $C1(s', i) \wedge C2(s', i) \wedge C3(s', i)$. Let $r = s'.round(i)$, $v = s'.round(i)$. By Lemma 6.12, $s.value(i) = v$, $s.round(i) = r$, and $C1(s, i) \wedge C2(s, i) \wedge C3(s, i)$. Since $I(s)$ is true, we also have $Ca(s, i) \wedge Cb(s, i) \wedge Cc(s, i)$. We need to show $Ca(s', i) \wedge Cb(s', i) \wedge Cc(s', i)$.

To show $Ca(s', i)$ it is enough to show that $s'.rounds[k]_j \geq r \Rightarrow s'.values[k]_j = v$. From the transition relation of AP, $s'.rounds[k]_j = s.round(k)$ and $s'.values[k]_j = s.value(k)$. Thus, $Cb(s, i)$ suffices.

$Cb(s', i)$ follows trivially from $Cb(s, i)$ since none of the relevant state components change.

For $Cc(s', i)$, suppose that $j \in s'.obs_i$, $s'.round(j) = r - 1$, $s'.value(j) \neq v$. Observe that $i \neq j$ since $s.round(i) = r$ and thus $s'.round(i) \neq r - 1$. The terms $s.\mathit{fill\text{-}max\text{-}round}_j$ and $s'.\mathit{fill\text{-}max\text{-}round}_j$ differ only in the use of $round(k)$ and $rounds[k]_j$. The transition relation of AP ensures the equality of the two terms above. Thus, $Cc(s', i)$ follows from $Cc(s, i)$.

3. For some j, $a = check1_j$ or $a = check2_j$ or $a = \mathit{return\text{-}flip}(v', r')_i$, $v' \in \{0, 1\}$, $r' > 0$.

Consider a processes i such that $C1(s', i) \wedge C2(s', i) \wedge C3(s', i)$. Let $r = s'.round(i)$, $v = s'.round(i)$. Observe that, by Invariants 6.5 and 6.10, $s.round(j) = s.rounds[j]_j$. Furthermore, observe that for all processes l,

$$s'.rounds_l = s.rounds_l \wedge s'.values_l = s.values_l \wedge s'.obs_l \subseteq s.obs_l. \tag{14}$$

If $s'.pc_j \in \{decide, flip\}$, then $I(s')$ follows trivially from $I(s)$ since none of the relevant state components change. Thus, we consider only the case where $s'.pc_j \neq decide$. In particular, $s'.obs_j = \emptyset$.

If $i = j$ (and $s'.pc_i \notin \{decide, flip\}$), then from $s'.obs_i = \emptyset$ we get $Cc(s', i)$. Furthermore, $s'.obs_i = \emptyset$ and $C2(s', i)$ imply $s'.agree(r, v)$, and thus $Cb(s', i)$ is true. From $s'.obs_i = \emptyset$ and $C3(s', i)$, we derive $s'.max\text{-}round \leq r$. This means that the conditions of Invariant 6.11 are satisfied, and thus $Ca(s', i)$ is true.

If $i \neq j$ (and $s'.pc_i \notin \{decide, flip\}$), then Lemma 6.12 implies that $s.value(i) = v$ and $s.round(i) = r$ and $C1(s, i) \wedge C2(s, i) \wedge C3(s, i)$. Since $I(s)$ is true, we have $Ca(s, i) \wedge Cb(s, i) \wedge Cc(s, i)$. Equation (14) and $Ca(s, i)$ imply $Ca(s', i)$. Since $s'.round(i) = r$, from $s'.obs_j = \emptyset$ we derive $s'.fill\text{-}max\text{-}round_j \geq r$, and thus $Cc(s', i)$ follows from $Cc(s, i)$. To show $Cb(s', i)$ we distinguish the following cases.

(a) $s.round(j) \geq r$.

By Invariant 6.5, $s.obs_j = \{1, \ldots, n\}$, and thus, by Invariant 6.10, $s.rounds[j]_j = s.round(j)$. From $Ca(s, i)$, since $s.obs_j = \{1, \ldots, n\}$, and since $s.round(j) \geq r$, we derive $s.obs\text{-}leader\text{-}agree(v)_j$. By Invariant 6.7, $s.pc_j \neq wait$, and thus, from the transition relation of AP, $s'.value(j) = v$ and $s'.round(j) > r$. Therefore, $Cb(s', i)$ follows from $Cb(s, i)$.

(b) $s.round(j) = r - 1$ and $s.value(j) = \perp$.

By Invariant 6.5, $s.obs_j = \{1, \ldots, n\}$. If $j \in s'.obs_i$, then $Ca(s, i) \wedge Cc(s, i)$ implies $s.obs\text{-}leader\text{-}agree(v)_j$. By Invariant 6.7 and from the transition relation of AP, $s'.value(j) = v$ and $s'.round(j) = r$. Therefore $Cb(s', i)$ follows from $Cb(s, i)$. If $j \notin s'.obs_i$, then from $C2(s', i)$, $s'.value(j) = v$. Thus, $Cb(s', i)$ follows from $Cb(s, i)$.

(c) $s.round(j) = r - 1$ and $s.value(j) \neq \perp$.

By Invariant 6.5, $s.obs_j = \{1, \ldots, n\}$, and by Invariant 6.6, $a = check1$. If $j \notin s'.obs_i$, then $C2(s', i)$ implies $\neg s.obs\text{-}leader\text{-}agree(\bar{v})_j$, since otherwise $s'.value(j)$ would be $\bar{v}$; if $j \in s'.obs_i$ and $s.value(j) = v$, then $Ca(s, i)$ and Invariant 6.10 imply $\neg s.obs\text{-}leader\text{-}agree(\bar{v})_j$; if $j \in s'.obs_i$ and $s.value(j) = \bar{v}$, then from $Cc(s, i)$ we derive $s.fill\text{-}max\text{-}round_j \geq r$, and thus, from $Ca(s, i)$, $s.obs\text{-}leader\text{-}agree(v)_j$.

Thus, in every case we have $\neg s.obs\text{-}leader\text{-}agree(\bar{v})_j$. If $s.obs\text{-}leader\text{-}agree(v)_j$, then from the transition relation of AP we have $s'.value(j) = v$ and $s'.round(j) = r$. Therefore, $Cb(s', i)$ follows from $Cb(s, i)$. If $\neg s.obs\text{-}leader\text{-}agree(v)_j$, then from the transition relation of AP we have $s'.value(j) = \perp$ and $s'.round(j) = r - 1$. Again, $Cb(s', i)$ follows from $Cb(s, i)$.

(d) $s.round(j) < r - 1$.

Since $s'.round(j) \leq r - 1$, $Cb(s', i)$ follows trivially from $Cb(s, i)$.

4. None of the previous cases hold.

$I(s)$ implies $I(s')$ since all the relevant components of s and s' stay unchanged. ■

Proof of Invariant 6.1
Follows directly from Invariant 6.3. ■

[[[**RS.** ADDED THIS PROOF.]]]

7 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 existing techniques for ordinary nondeterministic systems and confine the probabilistic arguments to a very limited section of the analysis. In this way we can also point out very precisely what is the essential role of probability within the protocol we analyze. The results of this section are integrated with probabilistic arguments in Section 8.

For each round r, let CF_r be a coin flipping protocol, that is, a probabilistic automaton with the interface of a coin flipper of Figure 2. Define AH (Aspnes-Herlihy) to be $AP \parallel (\|_{r \geq 1} CF_r)$.

For each finite execution fragment α of AH, define

$$\phi_{MaxRound}(\alpha) \stackrel{\triangle}{=} lstate(\alpha).max\text{-}round - fstate(\alpha).max\text{-}round,$$

where *max-round* is a function that gives the maximum round number among all the processes. Since the round number of each process is monotonic non-decreasing, it is immediate to verify that $\phi_{MaxRound}$ is a complexity measure. Define the following sets of states.

$\mathcal{R}$ the set of reachable states of AH such that $\exists_i pc_i \notin \{init, nil\}$;

$\mathcal{D}$ the set of reachable states of AH such that $\forall_i (pc_i \in \{init, nil\})$.

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

$\mathcal{F}_0$ the set of states of $\mathcal{R}$ where there exists a round r and a process l such that $round(l) = r$, $value(l) = 0$, $obs_l = \emptyset$, and for all processes $j \neq l$, $round(j) < r$;

$\mathcal{F}_1$ the set of states of $\mathcal{R}$ where there exists a round r and a process l such that $round(l) = r$, $value(l) = 1$, $obs_l = \emptyset$, and for all processes $j \neq l$, $round(j) < r$.

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

1. (Proposition 7.3) 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.

2. (Proposition 7.8) 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.

[[[**RS.** DEFINED RESPONSIVENESS AND (V,R)-GLOBALITY.]]] To state formally the two properties above we need to define the meaning of the sentences "all invocations to the coin flippers on non-failing ports get responses", and "all invocations to CF_r get only response v", which we identify with the concepts of *responsivness* and (v,r)-*globality*, respectively.

Definition 7.1 A port i is non-failing in an execution fragment α of AH or of CF_r if action $stop_i$ does not occur in α.

An invocation to CF_r from process i is *pending* in a reachable state s of CF_r if there is an execution α of CF_r, ending in state s, such that in α port i is non-failing, there is at least one occurrence of action *start-flip*$(r)_i$, and the last occurrence of *start-flip*$(r)_i$ is not followed by any action of the form *return-flip*$(v,r)_i$.

An execution fragment α of CF_r is *responsive* if, for each decomposition $\alpha_1 \frown \alpha_2$ of α the following holds: if in $fstate(\alpha_2)$ there is a pending request of process i to CF_r, then in α_2 either action $stop_i$ occurs, or action *return-flip*$(v,r)_i$ occurs for some $v \in \{0,1\}$. An execution fragment α of AH is *responsive* if, for each $r > 0$ $\alpha \lceil CF_r$ is responsive.

An execution fragment α of CF_r is v-*global* iff for each action of the form *return-flip*$(v',r)_i$ that occurs in α, $v' = v$. An execution fragment α of AH is (v,r)-*global* iff $\alpha \lceil CF_r$ is v-global. ■

Remark 7.1 The definition of pending request may appear rather cumbersome, since we could state it just in terms of the components of a state of CF_r. The problem is that CF_r is not specified yet, and thus we connot refer to its state components: we can refer only to the interactions that CF_r has with its external environment. ■

Statement 1 is almost trivial and states that within one round some process moves first to a new round or all processes terminate. Statement 2 is the key result of this section. It states that if the maximum round is r and the process at round r has value v, then the system quiesces within two rounds if CF_r behaves like a *global coin* flipper. We start with Statement 1, which requires a trivial preliminary lemma.

Lemma 7.2 *Let α be a fair execution fragment of AH that starts from a state of $\mathcal{R}$, and assume that α is responsive. [[[***RS.** RESPONSIVE*]]] Then in α either a state from $\mathcal{D}$ is reached, or max-round grows unboundedly.*

Proof. Follows directly from the fact that all processes perform finitely many operations in every round. ■

Proposition 7.3 *Let s_0 be a state of $\mathcal{R}$, and let α be a fair execution fragment of AH that starts from state s_0. Suppose that α is responsive. [[[***RS.** RESPONSIVE*]]] Then in α a state of $\mathcal{F}_0 \cup \mathcal{F}_1 \cup \mathcal{D}$ is reached within one round. That is, $\alpha = \alpha_1 \frown \alpha_2$ such that $lstate(\alpha_1) \in \mathcal{F}_0 \cup \mathcal{F}_1 \cup \mathcal{D}$ and $\phi_{MaxRound}(\alpha_1) \leq 1$.*

Proof. If $\mathcal{D}$ is not reached, then, by Lemma 7.2, *max-round* grows unboundedly. Thus, some process will be the first process to reach round $s_0.max\text{-}round + 1$. At that point a state from $\mathcal{F}_0 \cup \mathcal{F}_1$ is reached. ■

This proves Statement 1. For Statement 2 we need to prove some preliminary invariants. The first invariant is an immediate consequence of the fact that a process has a correct view of itself whenever it has observed itself.

Invariant 7.4 *Let i be a process. Then, for each reachable state of AH,*

$$fill\text{-}max\text{-}round_i \geq round(i).$$

Proof. Straightforward inductive argument. ■

The second invariant states that the round of each process is monotonically increasing and that a process cannot prefer both values 0 and 1 in the same round.

Invariant 7.5 *Let α be an execution fragment of AH, and let $s_0 = fstate(\alpha)$ be reachable in AH. Let l be a process, $r = s_0.round(l)$, and $v = s_0.value(l)$. If $v \neq \perp$, then for each state of α,*

$$round(l) \geq r \wedge (round(l) = r \Rightarrow value(l) \neq \bar{v}).$$

Proof. Straightforward inductive argument. ■

The third invariant is more technical. The important part is the second condition, which states that all processes observe agreement on value v from round $r+1$ provided that the coin flipper for round r always returns v, [[[**RS.** REPLACED GLOBAL]]] that at the beginning there is exactly one process at round r, and that the process at round r prefers value v. The other two conditions are necessary to carry out the inductive proof.

Invariant 7.6 *Let α be an execution fragment of AH whose first state s_0 is a state of $\mathcal{F}_v$. Let $r = s_0.max\text{-}round$, and let l be the (unique) process that satisfies $s_0.round(l) = r$. Suppose that α is (v, r)-global. [[[***RS.*** V,R GLOBAL]]] Then, for each state of α,*

1. $\forall_j(round(j) = r \Rightarrow \neg fill\text{-}leader\text{-}agree(\bar{v})_j)$
2. $\forall_j fill\text{-}agree(r+1, v)_j$.
3. $agree(r+1, v)$.

Proof. For notational convenience let $I(s)$ denote the whole invariant. State s_0 satisfies Conditions 2 and 3 trivially since $s_0.max\text{-}round < r + 1$. For Condition 1, since process l is the only process at round r, and since $s_0.value(l) = v$ and $s_0.obs_l = \emptyset$, it cannot be the case that $s_0.fill\text{-}leader\text{-}agree(\bar{v})_l$. For the inductive step we consider a subsequence sas' of α and we distinguish cases based on a.

1. $a = init(v')_i$ for some i.

   If $r > 1$, then none of the conditions of $I(s)$ are affected. If $r = 1$, then Conditions 2 and 3 are not affected as well. Consider a generic process j such that $s'.round(j) = r$. If $j = i$, then since $s'.obs_j = \emptyset$, Invariant 7.5 and Condition 3 for s' ensure that $\neg fill\text{-}leader\text{-}agree(\bar{v})_j$. If $j \neq i$, then, since Condition 1 holds in s, there is some process $k \neq i$ that is a leader with value different from $\bar{v}$ in the filled vector of process j. We know that $k \neq i$ because, by Invariant 7.4, $s.fill\text{-}max\text{-}round_j \geq r$, and thus process i could not affect Condition 1 in s. The k^{th} entry of the filled vector of j is not affected during the transition from s to s', and thus Condition 1 is preserved.

2. $a = read1(k)_i$ or $a = read2(k)_i$ for some i and k.

   In this case Condition 3 is not affected. Thus, we need to deal only with Conditions 1 and 2, which are affected only for process i. In particular, Conditions 1 and 2 differ in s and s' for the use of $(round(k), value(k))$ and $(rounds[k]_i, values[k]_i)$, respectively. The transition relation of AP ensures the equality of the terms above, and thus the preservation of Conditions 1 and 2.

3. $a = check1_i$ or $a = check2_i$ or $a = return\text{-}flip(v', r')_i$ for some i.

   We consider only the case where $r' = round(i)$, since otherwise nothing changes during the transition from s to s'. We distinguish the following cases.

   (a) $s.round(i) < r - 1$.
   In this case $I(s')$ follows trivially from $I(s)$ since none of the conditions are affected.

   (b) $s.round(i) = r - 1$.
   Conditions 2 and 3 are not affected. If $s'.round(i) = r - 1$, then Condition 1 is not affected as well. Otherwise, $s'.obs_i = \emptyset$. Observe also that $i \neq l$. Thus, Condition 1 follows from Condition 1 for s and by Invariant 7.5.

(c) $s.round(i) = r$.

If $s'.round(i) = r$, then Conditions 2 and 3 are not affected. For Condition 1, if $s'.pc_i = decide$, then Condition 1 is not affected; otherwise, $s'.value(i) = \perp$ and $s'.obs_i = \emptyset$. Thus, Condition 1 follows from Condition 1 for s and from Condition 3.

If $s'.round(i) = r + 1$, then Condition 1 and the transition relation of $AP(v, r)$ ensure that $s'.value(i) = v$. Thus, Conditions 1, 2 and 3 are all preserved.

(d) $s.round(i) > r$.

From Condition 2 on s, either process i decides on v, or a new round is reached with preference v. In both cases Conditions 1, 2 and 3 are preserved.

4. None of the previous cases hold.

$I(s')$ follows trivially from $I(s)$ since none of the relevant state components change. ■

Finally, we can show that from $\mathcal{F}_v$ the maximum round of the processes does not grow by more than 2 provided that the coin flipper at the maximum round always returns v.

Invariant 7.7 *Let α be an execution fragment of AH whose first state s_0 is a state of $\mathcal{F}_v$. Let $r = s_0.max\text{-}round$. Suppose that α is (v, r)-global. [[[***RS.** V,R GLOBAL*]]] Then, for each state of α, and for each process j,*

$$round(j) \leq r + 2.$$

Proof. First observe that α satisfies the conditions of Invariant 7.6, which means that Invariant 7.6 is satisfied by all the states of α.

All the cases for the proof are straightforward except for the case where a transition $(s, check1_j, s')$ occurs and $s.round(j) = r + 2$. In such case, from Condition 2 of Invariant 7.6, $s.fill\text{-}agree(r + 1, v)_j$. Since $s.obs_j = \{1, \ldots, n\}$, we derive that $s.obs\text{-}agree(r + 1, v)$, and thus process j sets pc_j to *decide* without reaching round $r + 3$. Observe that $check2_j$ cannot occur when $round(j) = r + 2$ since in such case $value(j) = \perp$ and Invariant 7.6 would be violated. ■

Proposition 7.8 *Let α be a fair execution fragment of AH whose first state s_0 is a state of $\mathcal{F}_v$. Let $r = s_0.max\text{-}round$. Suppose that α is responsive and (v, r)-global. [[[***RS.** RESPONSIVE*]]] Then in α a state from $\mathcal{D}$ is reached within two rounds. That is, $\alpha = \alpha_1 \frown \alpha_2$ where $lstate(\alpha_1) \in \mathcal{D}$ and $\phi_{MaxRound}(\alpha_1) \leq 2$.*

Proof. Suppose that $\mathcal{D}$ is not reached in α. Then, by Lemma 7.2, some process eventually reaches round $r + 3$, contradicting Invariant 7.7. Therefore, in α a state from $\mathcal{D}$ is reached. Furthermore, by Invariant 7.7, a state from $\mathcal{D}$ is reached within two rounds. ■

8 Probabilistic Progress Properties

Suppose that each coin flipping protocol CF_r satisfies the following properties. [[[**RS.** CHANGED STATEMENTS]]]

C1 For each fair probabilistic execution fragment of CF_r that starts with a reachable state of CF_r, the probability of the executions that are responsive is 1.

C2 For each fair probabilistic execution of CF_r, and each value $v \in \{0, 1\}$, the probability of the executions that are responsive and v-global is at least p, where p is a real number such that $0 < p \leq 1$.

In this section we show that under Conditions **C1** and **C2** for every CF_r, AH guarantees progress within expected $O(1/p)$ rounds. That is, we prove the following proposition.

Proposition 8.1 *If each coin flipping protocol CF_r satisfies properties* **C1** *and* **C2**, *then in AH, starting from any state of $\mathcal{R}$ and under any fair scheduler, a state from $\mathcal{D}$ is reached within at most $O(1/p)$ expected rounds.*

Thus, we need to show only that it is possible to build distributed implementations of the coin flippers that satisfy **C1** and **C2** with a suitable value for p. We build the implementations in Sections 9 and 10. [[[**RS.** ADDED LAST SENTENCE.]]]

Remark 8.1 Observe that property **C1** refers to probabilistic execution fragments, while Property **C2** refers to probabilistic executions. This distinction is important. Property **C1** states that a coin flipper gives responses with probability 1 from any arbitrary point in its computation; Property **C2** guarantees that with probability p a specific value is always returned, but only if we observe the coin flipper from the beginning. **C2** is not true for an arbitrary probabilistic execution fragment: if we consider a fragment that begins in a state where two processes are about to return two different values, then all processes return the same value with probability 0. ■

We now turn to the proof of Proposition 8.1. The main statement that we use is

$$\mathcal{R} \xrightarrow[p]{\phi_{MaxRound} \leq 3} \mathcal{D}. \tag{15}$$

To prove Statement (15) we prove two intermediate statements:

$$\mathcal{R} \xrightarrow[1]{\phi_{MaxRound} \leq 1} \mathcal{F}_0 \cup \mathcal{F}_1 \cup \mathcal{D}, \tag{16}$$

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

$$\mathcal{F}_v \xrightarrow[p]{\phi_{MaxRound} \leq 2} \mathcal{D}. \tag{17}$$

The proofs of Statements (16) and (17) rely on Propositions 7.3 and 7.8 and on the probabilistic properties of the coin flipping protocols. In particular, the first statement relies on the fact that the coin flippers respond, which occurs with probability 1 (**C1**), and the second statement relies on the fact that some specific coin flipper always returns a specific value v, which is the case with probability at least p (**C2**). [[[**RS.** MODIFIED PARAGRAPH.]]]

Proposition 8.2 *Assuming that the coin flippers in AH satisfy* **C1**,

$$\mathcal{R} \xrightarrow[1]{\phi_{MaxRound} \leq 1} \mathcal{F}_1 \cup \mathcal{F}_0 \cup \mathcal{D}. \tag{18}$$

Proof. Let H be a probabilistic execution fragment of AH that starts from a state of $\mathcal{R}$. Let Θ be the set of executions of Ω_H where each invocation to any coin flipper on a non-failing port gets a response. By Proposition 7.3, in each execution of Θ a state from $\mathcal{F}_1 \cup \mathcal{F}_0 \cup \mathcal{D}$ is reached within one round. Thus, it is sufficient to show that $P_H[\Theta] = 1$. Let, for each $i \geq 1$, Θ_i be the set of executions of Ω_H where each invocation to CF_i on a non-failing port gets a response. Then $\Theta = \cap_{i \geq 1} \Theta_i$. Observe that, by definition, Θ_i is the inverse image under projection of the set of executions of $\Omega_{H \lceil CF_i}$ where each invocation on a non-failing port gets a response. From **C1**, for each i, $P_{H \lceil CF_i}[\Theta_i \lceil CF_i] = 1$, and thus, by Proposition 2.3, $P_H[\Theta_i] = 1$. Therefore, $P_H[\Theta] = 1$ since any countable intersection of probability 1 events has probability 1. ■

Proposition 8.3 *Assuming that the coin flippers in AH satisfy* **C1** *and* **C2**,

$$\mathcal{F}_v \xrightarrow[p]{\phi_{MaxRound} \leq 2} \mathcal{D}. \tag{19}$$

Proof. Let H be a probabilistic execution fragment of AH that starts from a state s_0 of $\mathcal{F}_v$, and let $r = s_0.\textit{max-round}$. Let Θ be the set of executions of Ω_H where each invocation to any coin flipper on a non-failing port gets a response and where each response of CF_r has value v. By Proposition 7.8, in each execution of Θ a state from $\mathcal{D}$ is reached within two rounds. Thus, it is sufficient to show that $P_H[\Theta] \geq p$. Let, for each $i \geq 1$, Θ_i be the set of executions of Ω_H where each invocation to CF_i on a non-failing port gets a response. Furthermore, let Θ'_r be the set of executions of Ω_H where no response of CF_r has value $\bar{v}$. Then, $\Theta = (\cap_{i \geq 1} \Theta_i) \cap \Theta'_r$. From **C1**, for each i, $P_{H \lceil CF_i}[\Theta_i \lceil CF_i] = 1$, and thus, by Proposition 2.3, $P_H[\Theta_i] = 1$. Since $s_0 \in \mathcal{F}_v$ and $r = s_0.\textit{max-round}$, $H \lceil CF_r$ is a probabilistic execution of CF_r (the start state of $H \lceil CF_r$ is a start state of CF_r), and thus property **C2** can be applied. From **C2**, $P_{H \lceil CF_r}[\Theta'_r \lceil CF_r] \geq p$, and thus, by Proposition 2.3, $P_H[\Theta'_r] \geq p$. Therefore, $P_H[\Theta] \geq p$

since any countable intersection of probability 1 events has probability 1 and the intersection of a probability 1 event with an event with probability p has probability at least p. ■

Proof of Proposition 8.1. By Proposition 2.8, Statements (16) and (17) can be combined to lead to Statement (15).

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 2.9 we derive that within at most expected $4/p$ rounds a state from $\mathcal{D}$ is reached under any fair scheduler. ■

9 The Coin Flipping Protocol

We are left to show that it is possible to build a distributed coin flipping protocol with the properties **C1** and **C2** stated in Section 8, where by a distributed protocol we mean a protocol where processes interact through single-writer multiple-reader shared variables only.

In this section we build an almost distributed version of the coin flipping protocol where processes interact through a multiple-writer multiple-reader shared register; in Section 10 we refine the protocol of this section to yield a distributed protocol. The protocol is based on random walks and satisfies properties **C1** and **C2** with a sufficiently high probability p that is independent of n. [[[**RS.** REWORDED PARAGRAPH.]]]

9.1 The Code for the Protocol

We represent the coin flipping protocol by letting an automaton DCN_r (Distributed CoiN) [[[**RS.** EXPLAINED NAMING]]] interact with a centralized counter CT_r (CounTer), that is, $CF_r = Hide_I(DCN_r \parallel CT_r)$, where I is the set of actions used for the interaction between DCN_r and CT_r, and $Hide_I$ is an operator that transforms the actions of I from external to internal. [[[**RS.** EXPLAINED THE HIDING OPERATOR]]]. Figure 3 shows the structure of the coin flipping protocol. In this Section, DCN_r is distributed while CT_r is composed of n processes that receive requests from DCN_r and read/update a single shared variable: the details of the distributed implementation of a shared counter are not necessary for any properties of the coin flipping protocol. The distributed version of the shared counter is presented in Section 10.

Since the protocols for DCN_r and CT_r are the same for any round r, we drop the subscript r from our notation. Table 3 gives the state variables of DCN; Table 4 gives the transition relation of 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, and if the value read is beyond the barriers $\pm Kn$, where K is a fixed constant, then the process returns. The protocol described in Table 4 is slightly different from the protocol described in [3]: once a coin flip is requested, our protocol checks *counter* before flipping a coin, while the protocol of [3] starts immediately by flipping a coin. Our protocol improves the protocol of [3] in that

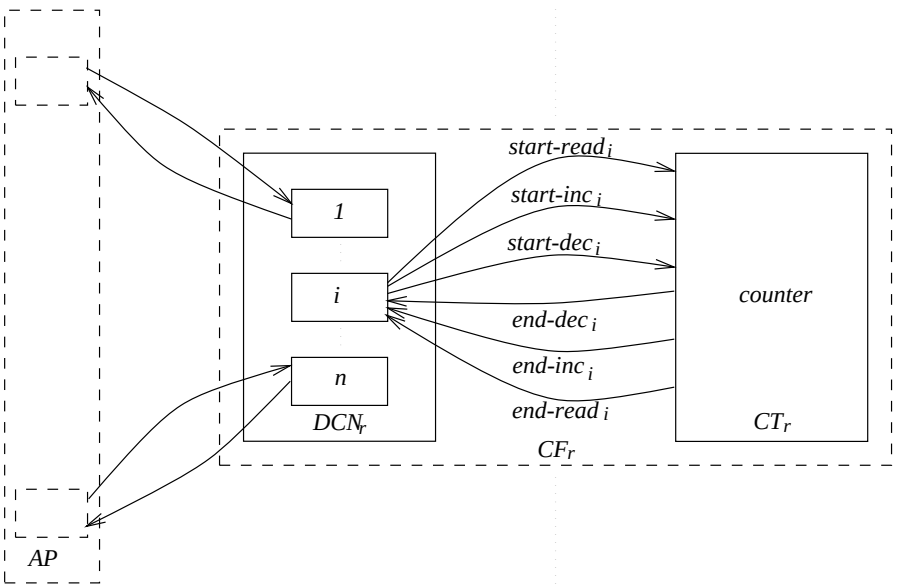

Figure 3: The structure of the coin flipping protocol.

properties **C1** and **C2** are satisfied even in the presence of multiple requests on the same port. This improvement is not essential for the correctness of the protocol of [3], since the protocol guarantees that there is at most one request at each port; however, our improvement simplifies the proof slightly in that we do not have to prove explicitly that there is at most one request at each port.

Table 5 gives the state variables of the shared counter CT; Table 6 gives the actions and transition relation of CT. Informally, each process of CT receives requests that are handled by referring to a multiple-writer multiple-reader shared variable *counter*. Increment and decrement operations are performed by updating *counter* directly; read operations are implemented by first copying the value of *counter* to a multiple-writer single-reader variable *preread* and then, in a separate step, returning the value of *preread* to the environment. However, an update to *counter* may invalidate the value that a read operation is ready to return. This fact is expressed by the nondeterministic choice to reset any set of *preread* variables to $\perp$ whenever a process updates *counter*. Due to the way the *preread* variables are handled, 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. Essentially, our use of the *preread* variables is an abstraction of what the implementation of Section 10 actually does.

We now proceed with the analysis of CF. In particular, we show that with probability 1, all the invocations to CF on a non-failing port get an answer, and, for $v \in \{0, 1\}$, with probability at least $(K-1)/2K$ all the answers are v. The analysis is split into two parts: the first part deals with non-probabilistic properties, while the second part deals with probability.

Name	Values	Initially
Local state		
fpc	$\{nil, flip, inc, wait\text{-}inc, dec, wait\text{-}dec, read\text{-}counter, wait\text{-}counter, compare, return\text{-}flip_0, return\text{-}flip_1\}$	*nil*
stopped	*Bool*	*false*
local-counter	*int*	0

Table 3: The state variables of a process i in DCN.

9.2 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*. Function ϕ_{inc} and ϕ_{dec} correspond to functions $Heads_{Acts,\mathbf{S}}$ and $Tails_{Acts,\mathbf{S}}$ in Section 3.3; the difference $\phi_{inc}(\alpha) - \phi_{dec}(\alpha)$ corresponds to $Diff_{Acts,\mathbf{S}}(\alpha)$. Given a state s of CF, let $|s|_{inc}$ be the number of processes in s whose program counter of either DCN or CT is *inc*, and let $|s|_{dec}$ be the number of processes in s whose program counter of either DCN or CT is *dec*. Formally, let $S_{inc} = \{j \mid s.fpc_j = inc \vee s.cpc_j = inc\}$, the processes that are about to increment, and let $S_{dec} = \{j \mid s.fpc_j = dec \vee s.cpc_j = dec\}$, the processes that are about to decrement. Let $|s|_{inc} = |S_{inc}|$ and $|s|_{dec} = |S_{dec}|$. The following lemma states how *counter* and the actual number of coin flips giving *inc* and *dec* are related.

Lemma 9.1 *Let α be a finite execution of CF, and let $s = lstate(\alpha)$. Then,*

$$\phi_{inc}(\alpha) - \phi_{dec}(\alpha) = s.counter + |s|_{inc} - |s|_{dec}.$$

Proof. Straightforward induction on the length of α. ∎

Given a state s, let S_{below} (S_{above}) be the set of processes in s that have a pending request and either are up to flipping an elementary coin or are up to detecting that *counter* is below (above) the barrier Kn ($-Kn$). Let $|s|_{below}$ and $|s|_{above}$ denote the cardinality of S_{below} and S_{above}, respectively. Formally, S_{below} is the set of processes i such that either

1. $s.fpc_i = flip$, or

Actions and transitions of process i.

input $start\text{-}flip(r)_i$
Eff: if $fpc = nil \wedge \neg stopped$ then
$fpc \leftarrow read\text{-}counter$

output $start\text{-}read_i$
Pre: $fpc = read\text{-}counter$
Eff: $fpc \leftarrow wait\text{-}counter$

input $end\text{-}read(c)_i$
Eff: if $fpc = wait\text{-}counter$ then
$local\text{-}counter \leftarrow c$
$fpc \leftarrow compare$

internal $compare_i$
Pre: $fpc = compare$
Eff: if $local\text{-}counter \geq Kn$ then
$fpc \leftarrow return\text{-}flip_1$
elseif $local\text{-}counter \leq -Kn$ then
$fpc \leftarrow return\text{-}flip_0$
else
$fpc \leftarrow flip$

output $return\text{-}flip(v, r)_i$
Pre: $fpc = return\text{-}flip_v$
Eff: $fpc_i \leftarrow nil$

internal $flip(r)_i$
Pre: $fpc = flip$
Eff: $\Pr[fpc \leftarrow inc] = 1/2 \wedge$
$\Pr[fpc \leftarrow dec] = 1/2$

output $start\text{-}inc_i$
Pre: $fpc = inc$
Eff: $fpc \leftarrow wait\text{-}inc$

input $end\text{-}inc_i$
Eff: if $fpc = wait\text{-}inc$ then
$fpc \leftarrow read\text{-}counter$

output $start\text{-}dec_i$
Pre: $fpc = dec$
Eff: $fpc \leftarrow wait\text{-}dec$

input $end\text{-}dec_i$
Eff: if $fpc = wait\text{-}dec$ then
$fpc \leftarrow read\text{-}counter$

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

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

Table 4: The actions and transition relation of DCN.

2. $s.fpc_i = read\text{-}counter$ and $s.counter < Kn$, or
3. $s.fpc_i = compare$ and $s.local\text{-}counter_i < Kn$, or
4. $s.cpc_i = read\text{-}counter$ and either $s.preread_i < Kn$ or $s.counter < Kn$.

Similarly, S_{above} can be defined by replacing $< Kn$ with $> -Kn$. The following two lemmas state a key property for the analysis of the coin flipping protocol. We describe only Lemma 9.2 since Lemma 9.3 is symmetric. Suppose that a state is reached where the value of *counter* minus the number of processes that either are up to decrementing *counter* or are up to detecting that *counter* is below Kn is at least Kn. Then Lemma 9.2 states that this property continues to remain valid in the future. Roughly speaking, each process that reads *counter* terminates (does not flip nor update *counter* any more) because it observes a value that is at least Kn.

Name	Values	Initially
Local state		
cpc	$\{nil, wait, inc, end\text{-}inc, dec, end\text{-}dec, read\text{-}counter\}$	*wait*
stopped	*Bool*	*false*
Multiple-writer multiple-reader shared variables		
counter	*int*	0
Multiple-writer single-reader shared variables (process i reads)		
$preread(i)$	$int \cup \{\perp\}$	$\perp$

Table 5: The state variables of a process i in CT.

Lemma 9.2 *The following property is stable for CF, that is, it continues to be satisfied once it is satisfied.*

$$s.counter - |s|_{dec} - |s|_{below} \geq Kn. \tag{20}$$

Proof. Straightforward inductive argument. ■

Lemma 9.3 *The following property is stable for CF.*

$$s.counter + |s|_{inc} + |s|_{above} \leq -Kn. \tag{21}$$

Proof. Straightforward inductive argument. ■

A simple consequence of Lemmas 9.2 and 9.3 is that whenever the difference between the coin flips that give *inc* and the coin flips that give *dec* is beyond the barriers $\pm(K+1)n$, the value of *counter* is always beyond $\pm Kn$.

Lemma 9.4 *Let $\alpha = \alpha_1 \frown \alpha_2$ be an execution of CF such that $\phi_{inc}(\alpha_1) - \phi_{dec}(\alpha_1) = (K+1)n$. Then each state of α_2 satisfies counter $\geq Kn$.*

Proof. By Lemma 9.1, $\phi_{inc}(\alpha_1) - \phi_{dec}(\alpha_1) = s.counter + |s|_{inc} - |s|_{dec}$ where $s = lstate(\alpha_1) = fstate(\alpha_2)$, and thus $s.counter + |s|_{inc} - |s|_{dec} = (K+1)n$. By a simple algebraic manipulation,

Actions and transitions of process i.

input $start\text{-}inc_i$
Eff: if $cpc = wait$ then
$cpc \leftarrow inc$

internal inc_i
Pre: $cpc = inc$
Eff: $counter \leftarrow counter + 1$
$\forall_j preread(j) \leftarrow choose(preread(j), \perp)$
$cpc \leftarrow end\text{-}inc$

output $end\text{-}inc_i$
Pre: $cpc = end\text{-}inc$
Eff: $cpc \leftarrow wait$

input $start\text{-}dec_i$
Eff: if $cpc = wait$ then
$cpc \leftarrow dec$

internal dec_i
Pre: $cpc = dec$
Eff: $counter \leftarrow counter - 1$
$\forall_j preread(j) \leftarrow choose(preread(j), \perp)$
$cpc \leftarrow end\text{-}dec$

output $end\text{-}dec_i$
Pre: $cpc = end\text{-}dec$
Eff: $cpc \leftarrow wait$

input $start\text{-}read_i$
Eff: if $cpc = wait$ then
$cpc \leftarrow read\text{-}counter$

internal $read_i$
Pre: $cpc = read\text{-}counter$
$preread(i) = \perp$
Eff: $preread(i) \leftarrow counter$

output $end\text{-}read(c)_i$
Pre: $cpc = read\text{-}counter$
$preread(i) = c \neq \perp$
Eff: $cpc \leftarrow wait$
$preread(i) \leftarrow \perp$

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

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

Table 6: The actions and transition relation of CT.

$s.counter - |s|_{dec} - |s|_{below} = s.counter + |s|_{inc} - |s|_{dec} - (|s|_{inc} + |s|_{below})$. Observe that, by definition, $S_{inc} \cap S_{below} = \emptyset$, and therefore $|s|_{inc} + |s|_{below} \leq n$. This means that $s.counter - |s|_{dec} - |s|_{below} \geq Kn$. By Lemma 9.2, each state s' of α_2 satisfies $s'.counter - |s'|_{dec} - |s'|_{below} \geq Kn$. Thus, each state of α_2 satisfies $counter \geq Kn$. ■

Lemma 9.5 *Let $\alpha = \alpha_1 \frown \alpha_2$ be an execution of CF such that $\phi_{inc}(\alpha_1) - \phi_{dec}(\alpha_1) = -(K+1)n$. Then each state of α_2 satisfies $counter \leq -Kn$.*

Proof. Symmetric to the proof of Lemma 9.4. ■

Lemma 9.6 *Let α be an 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 α is 1-global. [[[**RS.** CHANGED STATEMENT]]]*

Proof. Since $\alpha \in \mathbf{Top}[-(K-1)n, (K+1)n, 0](H)$, either each prefix α' of α satisfies $-(K-1)n < \phi_{inc}(\alpha') - \phi_{dec}(\alpha') < (K+1)n$, or $\alpha = \alpha_1 \frown \alpha_2$ where $\phi_{inc}(\alpha_1) - \phi_{dec}(\alpha_1) = (K+1)n$ and no prefix α_1' of α_1 satisfies $\phi_{inc}(\alpha_1') - \phi_{dec}(\alpha_1') \leq -(K-1)n$.

In the first case, by Lemma 9.1, no state of α satisfies *counter* $\leq -Kn$. In the second case, by Lemma 9.1, no state of α_1 satisfies *counter* $\leq -Kn$. Furthermore, by Lemma 9.4, each state of α_2 satisfies *counter* $\geq Kn$. Therefore, no state of α satisfies *counter* $\leq -Kn$. This means that in both cases no process returns value 0 in α. ■

Lemma 9.7 *Let α be an execution of CF, such that $\alpha \in$* **Bottom**$[-(K-1)n, (K+1)n, 0](H)$ *for some probabilistic execution H of CF. Then α is* 0*-global. [[[***RS.** CHANGED STATEMENT*]]]*

Proof. Symmetric to the proof of Lemma 9.6. ■

Lemma 9.8 *Let α be a fair execution of CF, such that $\alpha \in$* **Either**$[-(K+1)n, (K+1)n, 0](H)$ *for some probabilistic execution H of CF. Then α is responsive. [[[***RS.** CHANGED FORMULATION.*]]]*

Proof. If α contains finitely many flip actions, then eventually all the increment and decrement operations deriving from the flipping operations are completed or interrupted (the corresponding *end-inc* or *end-dec* actions occur or the corresponding processes fail). Thus, there is a point after which no more *inc* and *dec* operations are performed. Let α' be a suffix of α where no more flip, increment or decrement operations are performed. Then in α' none of the $preread_i$ variables is set to $\perp$ while action *end-read*$(c)_i$ is enabled, and thus all read operations on non-failing ports terminate eventually. At that point, since no more flips are performed in α', each process that completes a read operation returns a value.

If α contains infinitely many flip actions, then, since $\alpha \in$ **Either**$[-(K+1)n, (K+1)n, 0](H)$, $\alpha = \alpha_1 \frown \alpha_2$ such that $\phi_{inc}(\alpha_1) - \phi_{dec}(\alpha_1) = \pm(K+1)n$. Here we consider the case where $\phi_{inc}(\alpha_1) - \phi_{dec}(\alpha_1) = (K+1)n$; the other case is symmetric. By Lemma 9.4, each state of α_2 satisfies *counter* $\geq Kn$. Thus, each non-failing process returns a value once it reads *counter* (performing the read operation in α_2) since the value read is at least Kn. ■

Lemma 9.9 *Let α be a fair execution of CF, such that $\alpha \in$* **Top**$[-(K-1)n, (K+1)n, 0](H)$ *for some probabilistic execution H of CF. Then α is responsive and* 1*-global. [[[***RS.** REWORDED*]]]*

Proof. By Lemma 9.8, each invocation on a non-failing port gets a response. By Lemma 9.6 no invocation gets response 0. Hence, each invocation on a non-failing port gets response 1. ■

Lemma 9.10 *Let α be a fair execution of CF, such that $\alpha \in$* **Bottom**$[-(K-1)n, (K+1)n, 0](H)$ *for some probabilistic execution H of CF. Then α is responsive and* 0*-global.*

Proof. Symmetric to the proof of Lemma 9.9. ■

9.3 Probabilistic Analysis

In this short subsection we prove the probabilistic properties of the coin flipping protocol, that is, it guarantees properties **C1** (Proposition 9.11) and **C2** (Proposition 9.12). The proofs rely on the non-probabilistic properties proved in Section 9.2 and on the coin lemmas for symmetric random walks of Section 3.3.

Proposition 9.11 *The coin flipper CF satisfies* **C1**. *That is, for each fair probabilistic execution fragment of CF that starts with a reachable state of CF, the probability of the executions that are responsive is* 1.

Proof. Let H be a fair probabilistic execution fragment of CF that starts with a reachable state s of CF, and let α be a finite execution of CF such that $lstate(\alpha) = s$. Let $z = \phi_{inc}(\alpha) - \phi_{dec}(\alpha)$. If α' is an execution of the event $\mathbf{Either}[-(K+1)n, (K+1)n, z](H)$, then $\alpha \frown \alpha'$ is an execution of $\mathbf{Either}[-(K-1)n, (K+1)n, 0](H')$ for some fair probabilistic execution H' of CF, and by Lemma 9.8, every invocation to CF in $\alpha \frown \alpha'$ gets a response. From Definition 7.1, every invocation to CF in α' gets a response. By Theorem 3.11, $P_H[\mathbf{Either}[-(K+1)n, (K+1)n, z](H)] = 1$. This completes the proof. ■

Proposition 9.12 *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, the probability of the executions that are responsive and v-global is at least* $(K-1)/2K$. *[[[***RS.** FIXED STATEMENT*]]]*

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 9.9, every invocation to CF in α gets response 1. Furthermore, by Theorem 3.11, $P_H[\mathbf{Top}[-(K-1)n, (K+1)n, 0](H)] \geq (K-1)/2K$. This completes the proof. ■

10 Implementation of the Shared Counter

In this section we build an implementation of CT and we show that it can replace the abstract automaton CT in CF without compromising Propositions 9.11 and 9.12, that is, properties **C1** and **C2** with $p = (K-1)/2K$. In this way, using the coin flipping protocol with the new counter, we obtain a protocol for consensus that uses only single-writer multiple-reader shared variables.

The implementation of CT, which we denote by DCT (Distributed CounTer), [[[**RS.** ADDED NAME OF AUTOMATON]]] is an adaptation of an algorithm proposed by Lamport [8] for read/write registers. The state variable *counter* of CT is represented by n single-writer multiple-reader registers, one for each process, with two fields: a *num* field, which is incremented whenever the value of the register is changed, and a *val* field representing the

Name	Values	Initially
Local state		
cpc	$\{nil, wait, inc, end\text{-}inc, dec, end\text{-}dec, scan, read\text{-}counter\}$	*wait*
prescan	$array[1 \ldots n]$ of $int \times int$	array of $(0, 0)$
first	$array[1 \ldots n]$ of $int \times int$	array of $(0, 0)$
obs	set of $\{1, \ldots, n\}$	$\emptyset$
stopped	*Bool*	*false*
Single-writer multiple-reader shared variables		
$(num(i), val(i))$	$int \times int$	$(0, 0)$

Table 7: The state variables of a process i in DCT.

contribution of the corresponding process to the value of *counter*. The operations *inc* and *dec* on a process i are implemented by incrementing or decrementing the *val* register and incrementing the *num* register of process i. The operation *read-counter* is implemented by scanning the shared registers until two consecutive scans give the same value. Table 7 gives the state variables of DCT; Table 8 gives the transition relation of DCT.

We now verify that it is possible to replace DCT for CT in CF without compromising properties **C1** and **C2**. Let DCF (Distributed Coin Flipper) be defined as $Hide_I(DCN \| DCT)$, where I is the set of actions used for the interaction between DCN and DCT.

Observe that properties **C1** and **C2** are properties of the fair trace distributions of CF and DCF. Specifically, observe that responsivness and v-globality can be stated in terms of traces. Then, property **C1** can be stated as "in each fair trace distribution, the probability of the set of traces that are responsive is 1", and property **C2** can be stated as: "in each fair trace distribution, the probability of the set of traces that are responsive and v-global is at least p". [[[**RS.** EXCHANGED C1 AND C2.]]] [[[**RS.** FIXED STATEMENTS.]]] Thus, to show that DCF satisfies properties **C1** and **C2** it is sufficient to show that $ftdistrs(DCF) \subseteq ftdistrs(CF)$. For this purpose, by using Proposition 2.14, it is sufficient to build a refinement h from DCT to CT and show that h preserves the fair executions of DCT. Note that h is not probabilistic since DCT and CT are not probabilistic. That is, the properties that we need to show do not involve probability.

Proposition 10.1 *There is a refinement from DCT to CT that preserves the fair executions of DCT.*

Actions and transitions of process i.

input $start\text{-}inc_i$
Eff: if $cpc = wait$ then
$cpc \leftarrow inc$

internal inc_i
Pre: $cpc = inc$
Eff: $val(i) \leftarrow val(i) + 1$
$num(i) \leftarrow num(i) + 1$
$cpc \leftarrow end\text{-}inc$

output $end\text{-}inc_i$
Pre: $cpc = end\text{-}inc$
Eff: $cpc \leftarrow wait$

input $start\text{-}dec_i$
Eff: if $cpc = wait$ then
$cpc \leftarrow dec$

internal dec_i
Pre: $cpc = dec$
Eff: $val(i) \leftarrow val(i) - 1$
$num(i) \leftarrow num(i) + 1$
$cpc \leftarrow end\text{-}dec$

output $end\text{-}dec_i$
Pre: $cpc = end\text{-}dec$
Eff: $cpc \leftarrow wait$

input $start\text{-}read_i$
Eff: if $cpc = wait$ then
$cpc \leftarrow scan$
$obs \leftarrow \emptyset$

internal $scan(k)_i$
Pre: $cpc = scan$
$k \notin obs$
Eff: $scan[k] \leftarrow (counter(k), num(k))$
$obs \leftarrow obs \cup \{k\}$
if $obs = \{1, \ldots, n\}$ then
if $\neg first \wedge (prescan = scan)$ then
$first \leftarrow true$
$counter \leftarrow \sum_{j=1}^{n} scan_i[j].val$
$cpc \leftarrow read\text{-}counter$
else
$prescan \leftarrow scan$
$first \leftarrow false$

output $end\text{-}read(c)_i$
Pre: $cpc = read\text{-}counter$
$c = \sum_{j=1}^{n} scan[j].val$
Eff: $cpc \leftarrow wait$

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

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

Table 8: The actions and transition relation of DCT.

Proof. The refinement keeps the *preread* variables different from $\perp$ whenever the first scan has occurred and no increment or decrement operations have done anything that would make the first and second scans differ. Formally, $h(s) = s'$ where, for each process i,

$$s'.cpc_i = \begin{cases} read\text{-}counter & \text{if } s.cpc_i = scan \\ s.cpc_i & \text{otherwise,} \end{cases}$$

$$s'.counter = \sum_j val(j)$$

$$s'.preread_i \quad = \quad \begin{cases} c & \text{if } \neg s.first_i \text{ and } c = \sum_j s.prescan[j]_i \\ & \text{and } s.cpc_i \in \{scan, read\text{-}counter\} \\ & \text{and } \forall_j (j \in obs_i \Rightarrow prescan[j]_i = scan[j]_i) \\ & \text{and } \forall_j (j \notin obs_i \Rightarrow prescan[j]_i = (val(j), num(j))) \\ \bot & \text{otherwise.} \end{cases}$$

It is straightforward to check that h is a refinement mapping.

Consider now a fair execution α_1 of DCT. From the execution correspondence theorem there is an execution α_2 of CT such that $(\alpha_1, \alpha_2) \in h$. Suppose by contradiction that α_2 is not fair. Then in α_2 there is a process i whose corresponding task is eventually continuously enabled but never performed. Observe that h^{-1} preserves the enabledness of each task of CT, and that in DCT it is not possible that for some task T there is an execution fragment with infinitely many internal actions from T and no external action from T. Thus, since $(\alpha_1, \alpha_2) \in h$, eventually in α_1 the task of process i is continuously enabled but never performed. This means that α_1 is not fair, a contradiction. ■

Theorem 10.2 *The coin flipper DCF satisfies properties* **C1** *and* **C2** *with* $p = (K-1)/2K$.

Proof. By Proposition 10.1, there is a refinement from DCT to CT that preserves the fair executions of DCT. By Proposition 2.14, $ftdistrs(DCF) \subseteq ftdistrs(CF)$. This completes the proof. ■

11 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.

Theorem 11.1 *Using the coin flippers of Sections 9 and 10, AH guarantees wait-free termination within a constant expected number of rounds, that is, 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 DCF of Sections 9 and 10 satisfy properties **C1** and **C2** with $p = (K-1)/2K$, where K is a constant (cf. Theorem 10.2 and Propositions 9.11 and 9.12). By Proposition 8.1, AH guarantees wait-free termination within at most $O(2K/(K-1))$ expected rounds, that is, within a constant expected number of rounds. ■

We analyze some implications of Theorem 11.1. In particular, the definition of $\mathcal{D}$ may appear rather counterintuitive, since reaching $\mathcal{D}$ does not necessarily mean deciding: it is possible to reach $\mathcal{D}$ by letting processes fail. However, Theorem 11.1 gives enough information to derive several different termination properties as the following corollary shows.

Corollary 11.2 *[[[**RS.** ITEMS REWORDED.]]] Let H be a fair probabilistic execution fragment of AH, and suppose that H starts from a reachable state s of AH. Then the following properties are satisfied by H.*

1. *If in s all processes are initialized already, then within a constant expected number of rounds all non-failing processes decide.*
2. *If in s there is at least one initialized and non-failed process, and if no new processes fail in H, then a decision is reached within a constant expected number of rounds.*

Proof. To reach $\mathcal{D}$ all initialized processes must either fail or decide. In the first case, since $\mathcal{D}$ is reached, all non-failed processes have decided. In the second case, since there is at least a non-failed initialized process, and since such process does not fail, such process decides. ■

12 Timing Analysis of the Algorithm

[[[**RS.** REWRITTEN THE MODELING PART. NOW IT SHOULD BE MUCH MORE CLEAR SINCE I CAN DEAL WITH TIME WITHOUT LEAVING THE FRAMEWORK OF I/O AUTOMATA.]]] In this section we prove an upper bound on the expected time it takes for all processes to terminate, starting from an arbitrary reachable state, once all processes have some minimum speed. For this purpose we augment the I/O automata of the previous sections paper so that time can be observed. Our augmentation resembles the patient construction of [6]. Note that we cannot regard the augmentation we present in this paper as the definition of a general timed probabilistic model. Our augmentation is the minimum machinery that is necessary for the time analysis of an asynchronous algorithm.

12.1 Modeling Time

In order to model time we add a special 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. We call the new actions *time-passage actions*. 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 ϕ_t as follows:

$$\phi_t(\alpha) = lstate(\alpha).now - fstate(\alpha).now\,.$$

It is straightforward to check that ϕ_t is a complexity measure. 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 there is no task T of M and no decomposition $\alpha_1 \frown \alpha_2 \frown \alpha_3$ of α such that $\phi_t(\alpha_2) > 1$, all the states of α_2 enable T, and no action from T occurs in α_2. That is, α is well-timed if each task does not remain enabled for more than one time unit without being performed.

All the properties that we have studied in the previous sections are still valid for our augmented automata, since they are not affected by the presence of the *.now* component and of the new input actions. It is simple to observe that if we remove the time-passage transitions from a fair execution of an augmented automaton we obtain a fair execution of the non-augmented automaton.

In the rest of this section we strenghten the properties of the previous sections by showing that, under the assumption of well-timedness, the algorithm of Aspnes and Herlihy terminates within an expected polynomial time. That is, if from a certain point each processor has some minimum speed, then the algorithm of Aspnes and Herlihy guarantees termination within an expected polynomial time.

12.2 Preliminary Definitions

Before presenting the timing analysis we give some preliminary definitions. Recall that, for each $r > 0$, DCF_r denotes $Hide_I(DCN_r \parallel DCT_r)$, where I is the set of actions used for the interaction between DCN_r and DCT_r. That is, DCF_r is the result of substituting DCT_r for CT_r in CF_r. Let DAH (Distributed Aspens-Herlihy) denote $AP \parallel (\|_{r\geq 1} DCF_r)$. 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 $\phi_{id}(\alpha)$ denote the number of *inc* and *dec* events that occur in α. It is straightforward to check that $\phi_{flip,r}$, $\phi_{id,r}$ and ϕ_{id} are complexity measures. Observe that the following trivial result holds.

Lemma 12.1 *For each execution fragment α of DAH,*

1. $\phi_{id}(\alpha) = \sum_{r>0} \phi_{id,r}(\alpha)$, *and*
2. *for each* $r > 0$, $\phi_{id,r}(\alpha) = \phi_{id,r}(\alpha \lceil DCF_r)$. ■

12.3 Non-Probabilistic Properties of the Complexity Measures

In this section we study the relationship between the complexity measures $\phi_t, \phi_{id}, \phi_{flip}, \phi_{id,r}$, and $\phi_{flip,r}$ defined above. The first significant result of this section, Lemma 12.4, 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. We first prove a preliminary lemma, which provides a linear upper bound on the time a coin flipping protocol is active without any *inc*, *dec*, *return-flip* or *stop* action occurring. The preliminary lemma is first proved for a coin flipping protocol (cf. Lemma 12.2), and then proved for a coin flipping protocol within DAH. [[[**RS.** ADDED LAST SENTENCE.]]]

Lemma 12.2 *Let α be a fair, well-timed execution fragment of DCF_r, $r > 0$. Suppose that in $fstate(\alpha)$ there is at least one non-failed process with a pending start-flip(r) request. Then in α there is an occurrence of an action from $\{inc, dec, return\text{-}flip, stop\}$ within time $O(n)$.*

Proof. [[[**RS.** FIXED X.]]] Let X be $\{inc, dec, return\text{-}flip, stop\}$. Let i be a non-failed process with a pending *start-flip*(r) request in $fstate(\alpha)$, and suppose for the sake of contradiction that in α there is no occurrence of actions from X within time $3n + d$, where d is a sufficiently large constant. From the code of DCF_r, process i runs through a cycle where a read request is performed and an action from $\{inc, dec, return\text{-}flip\}$ occurs unless process i fails (action *stop*) occurs. Thus, one action from X occurs before completing a cycle. The maximum time necessary to complete a cycle is given by the time to complete a read request plus the time to check the result and perform the corresponding operations. The constant d accounts for the time necessary to complete all the operations except for the read request. Since no action from X occurs within time $3n + d$, a read request completes within time at most $3n$: in fact, within 3 scans of process i there are two consecutive scans that give the same result. Thus, within time $3n + d$ process i completes a cycle, which means that an action from X occurs, a contradiction. ■

Lemma 12.3 *Let α be a fair, well-timed execution fragment of DAH, and let $r > 0$. Suppose that in $fstate(\alpha)\lceil DCF_r$ there is at least one non-failed process with a pending start-flip(r) request. Then in α there is an occurrence of an action from $\{inc, dec, return\text{-}flip, stop\}$ within time $O(n)$.*

Proof. [[[**RS.** FIXED X.]]] Let X be $\{inc, dec, return\text{-}flip, stop\}$. By Lemma 12.2 in $\alpha\lceil DCF_r$ there is an occurrence of an action from X within time $c_1 n + c_2$ for appropriate constants c_1 and c_2. That is, $\alpha\lceil DCF_r = \alpha_1 \frown \alpha_2$ such that $\phi_t(\alpha_1) \leq c_1 n + c_2$ and an action from X occurs in α_1. Let α_1' be a prefix of α such that $\alpha_1 = \alpha_1'\lceil DCF_r$. Then, from the definition of projection, an action from X occurs in α_1', and from the definition of *.now* within parallel composition, $\phi_t(\alpha_1') = \phi_t(\alpha_1) \leq c_1 n + c_2$. This means that in α an action from X occurs within time $c_1 n + c_2$. ■

Lemma 12.4 *Let α be a well-timed execution fragment of DAH, and let $R = fstate(\alpha).max\text{-}round$. Suppose that all the states of α, with the possible exception of $lstate(\alpha)$ are active, that is, are states of $\mathcal{R}$. Then, $\phi_t(\alpha) \leq d_1 n^2(\phi_{MaxRound}(\alpha) + R) + d_2 n \phi_{id}(\alpha) + d_3 n^2$ for some constants d_1, d_2, and d_3.*

Proof. At each round each process performs a linear number of transitions outside the coin flipping protocol using time at most $c_1 n$ for some constant c_1. Divide α into two kinds of execution fragments: those where some active process is outside the coin flipping protocols, and those where no active process is outside the coin flipping protocols. The total time complexity of the first kind of execution fragments is upper bounded by $c_1 n^2(\phi_{MaxRound}(\alpha) + R)$, corresponding to the case where at each time there is exactly one process outside the coin flipping protocols. Consider now the second kind of execution fragments. Since each process returns at most once in each round and fails at most once overall, there are at most $\phi_{id}(\alpha) + n(\phi_{MaxRound}(\alpha) + R) + n$ events *inc*, *dec*, *return-flip* and *stop* in α. By Lemma 12.3, whenever some process is flipping, the maximum distance between two events of the kind *inc*, *dec*, *return-flip*, and *stop* is linear. Thus, the maximum time where some process is flipping in α (the time complexity of the second kind of execution fragments) is at most $c_1' n^2(\phi_{MaxRound}(\alpha) + R) + c_2 n \phi_{id}(\alpha) + c_3 n^2$ for some constants c_1', c_2, and c_3. Combining the two results, the time that elapses in α is at most $d_1 n^2(\phi_{MaxRound}(\alpha) + R) + d_2 n \phi_{id}(\alpha) + d_3 n^2$, where $d_1 = c_1 + c_1'$, $d_2 = c_2$, and $d_3 = c_3$. ■

The next two lemmas state basic properties of the coin flipping protocols. Lemma 12.5 derives from the fact that all the processes within a coin flipping protocol terminate once the shared counter reaches an absorbing barrier $(K+1)n$ or $-(K+1)n$. Essentially, once an absorbing barrier is reached, there are at most other n *flip* events, one for each process. Lemma 12.6 derives from the fact that each *inc* or *dec* event must be preceded by a *flip* event. If we start from an arbitrary reachable state, there could be some *inc* and *dec* events that occur without any preceeding *flip* event. However, the number of anomalous *inc* and *dec* events is at most n, that is, one for each process. [[[**RS.** MODIFIED PARAGRAPH.]]]

Lemma 12.5 *Let $\alpha = \alpha_1 \frown \alpha_2$ be a finite execution of DCF_r, and suppose that $|\phi_{inc}(\alpha_1) - \phi_{dec}(\alpha_1)| \geq (K+1)n$. Then $\phi_{flip,r}(\alpha_2) \leq n$.*

Proof. We consider the case where $\phi_{inc}(\alpha_1) - \phi_{dec}(\alpha_1) \geq (K+1)n$. The other case is symmetric. By Lemma 9.4, each state of α_2 satisfies $counter \geq Kn$, and thus each non-failing process returns 1 once it reads *counter* (performing the read operation in α_2) and checks its value. Each process can flip at most once in α_2 before starting a new read operation. Thus, the number of *flip* events that occur in α_2 is bound by n. ■

Lemma 12.6 *Let α be a finite execution fragment of DCF_r that starts from a reachable state. Then, $\phi_{id,r}(\alpha) \leq \phi_{flip,r}(\alpha) + n$.*

Proof. In $fstate(\alpha)$ there are at most n increment or decrement events that can be performed without first flipping a coin. ■

12.4 Expected Bound on Increment and Decrement Events

[[[**RS.** REPLACED OPERATIONS WITH EVENTS IN SEVERAL PLACES.]]] In this section we show an upper bound on the expected number of increment and decrement events that occur within a probabilistic execution of DAH. First, based on our results on random walks (cf. Proposition 3.12), we show in Lemma 12.7 an upper bound on the expected number of coin flips performed by a coin flipper. Then, in Lemma 12.8 we use this result together with our results about linear combinations of complexity measures (cf. Proposition 2.4) to derive an upper bound on the expected number of increment and decrement events performed by a coin flipper. Then, in Lemma 12.9 we use our compositionality results about complexity measures (cf. Proposition 2.6) to show that the bound of Lemma 12.8 is preserved by parallel composition. Finally, in Lemma 12.10 we use our result about phases of computations (cf. Proposition 2.5) to combine the result about the expected number of increment and decrement events of a coin flipper with our knowledge of the maximum expected number of coin flippers that may be invoked. This allows us to derive an upper bound on the expected total number of increment and decrement events during the consensus protocol.

Lemma 12.7 *Let H be a probabilistic execution fragment of DCF_r that starts from a reachable state of DCF_r, and let Θ be a full cut of H. Then $E_{\phi_{flip,r}}[H, \Theta] \leq (K+1)^2 n^2 + n$.*

Proof. Let s be the start state of H, and let α be a finite execution of DCF_r with $s = lstate(\alpha)$. Let $z = \phi_{inc}(\alpha) - \phi_{dec}(\alpha)$. If $|z| \geq (K+1)n$, then, by Lemma 12.5, for each $q \in \Theta$, $\phi_{flip,r}(q) \leq n$, and thus $E_{\phi_{flip,r}}[H, \Theta] \leq n$. If $|z| < (K+1)n$, then, by Proposition 3.12, $E_{\phi_{Acts,-(K+1)n,(K+1)n,z}}[H, \Theta] \leq -z^2 + (K+1)^2 n^2 \leq (K+1)^2 n^2$, that is, the event denoted by Θ is satisfied within expected $(K+1)^2 n^2$ *flip* events, truncating the count whenever an absorbing barrier $\pm(K+1)n$ is reached. Once an absorbing barrier is reached, by Lemma 12.5 there are at most n other flip events. Thus, for each state q of H, $\phi_{flip,r}(q) \leq \phi_{Acts,-(K+1)n,(K+1)n,z}(q) + n$. By Proposition 2.4, $E_{\phi_{flip,r}}[H, \Theta] \leq (K+1)^2 n^2 + n$. ■

Lemma 12.8 *Let H be a probabilistic execution fragment of DCF_r that starts from a reachable state of DCF_r, and let Θ be a full cut of H. Then $E_{\phi_{id,r}}[H, \Theta] \leq (K+1)^2 n^2 + 2n$.*

Proof. By Lemma 12.6, for each execution fragment of α of CF_r, $\phi_{id,r}(\alpha) \leq \phi_{flip,r}(\alpha) + n$. Then, by Proposition 2.4, $E_{\phi_{id,r}}[H, \Theta] \leq E_{\phi_{flip,r}}[H, \Theta] + n$. By Lemma 12.7, $E_{\phi_{flip,r}}[H, \Theta] \leq (K+1)^2 n^2 + n$. Thus, $E_{\phi_{id,r}}[H, \Theta] \leq (K+1)^2 n^2 + 2n$. ■

Lemma 12.9 *Let H be a probabilistic execution fragment of DAH that starts from a reachable state of DAH, and let Θ be a full cut of H. Then $E_{\phi_{id,r}}[H, \Theta] \leq (K+1)^2 n^2 + 2n$.*

Proof. Since $H \lceil DCF_r$ is a probabilistic execution fragment of DCF_r that starts from a reachable state of DCF_r, by Lemma 12.8, $E_{\phi_{id,r}}[H \lceil DCF_r, \Theta'] \leq (K+1)^2 n^2 + 2n$ for each full cut Θ' of $H \lceil DCF_r$. By Proposition 2.6, since by Lemma 12.1 for each execution fragment α of AH, $\phi_{id,r}(\alpha) = \phi_{id,r}(\alpha \lceil DCF_r)$, $E_{\phi_{id,r}}[H, \Theta] \leq (K+1)^2 n^2 + 2n$. ■

Lemma 12.10 *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 $E_{\phi_{id}}[H, \Theta] = O(Rn^2)$.*

Proof. If $R = 0$, then $\Theta = \{s\}$, and thus $E_{\phi_{id}}[H, \Theta] = 0 = O(Rn^2)$. For the rest of the proof assume that $R > 0$. Given a state q of H, we know that $\phi_{id}(q) = \phi_{id,1}(q) + \cdots + \phi_{id,R}(q) + \phi'(q)$, where $\phi'(q) = \sum_{r>0} \phi_{id,r+R}(q)$. For each $r > 0$, let Θ_r be the set of minimal states q of H such that $\phi_{MaxRound}(q) \geq r$. Then, for each $q \in \Theta_r$, $\phi_{id,r+R}(q) = 0$, and for each state q of H and each $r > \phi_{MaxRound}(q)$, $\phi_{id,r+R}(q) = 0$ (CF_{r+R} does not start until some process reaches round $r + R$). Furthermore, by Lemma 12.9, there is a constant $c = (K + 1)^2 n^2 + 2n$ such that for each probabilistic execution fragment H' of M, each full cut Θ' of H', and each $i > 0$, $E_{\phi_{id,i}}[H', \Theta'] \leq c$. Therefore, we are in the conditions to apply our result about phases of computation (cf. Proposition 2.5): each round is a phase, and the numbers of *inc* and *dec* events that occur within each round are the complexity measures for their corresponding round. Function $\phi_{MaxRound}$ is the measure of how many phases are started. By Proposition 2.5, $E_{\phi'}[H, \Theta] \leq cE_{\phi_{MaxRound}}[H, \Theta]$. By Theorem 11.1, $E_{\phi_{MaxRound}}[H, \Theta]$ is bound by a constant (independent of n). Therefore, $E_{\phi'}[H, \Theta] = O(n^2)$. Finally, since for each i, H, and Θ, $E_{\phi_{id,i}}[H, \Theta] = O(n^2)$, by Proposition 2.4, $E_{\phi_{id}}[H, \Theta] = O(Rn^2) + O(n^2) = O(Rn^2)$. ■

12.5 Expected Bound on Time

We are now ready to prove our main result, which is just a pasting together of the results obtained so far. Specifically, we show that starting from any reachable state of DAH, assuming well-timedness, a state from $\mathcal{D}$ is reached within expected time $O(Rn^3)$, where R is the maximum round of the processes at the starting state. Our result about reaching $\mathcal{D}$ implies directly several results about the termination properties of the consensus protocol of Aspnes and Herlihy (cf. Corollary 12.12).

Theorem 12.11 *Let H be a probabilistic fair, well-timed 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 $E_{\phi_t}[H, \Theta] = O(Rn^3)$.*

Proof. If $R = 0$, then $\Theta = \{s\}$, and thus $E_{\phi_t}[H, \Theta] = 0 = O(Rn^3)$. If $R > 0$, then, by Lemma 12.4, for each well-timed execution fragment α of DAH,

$$\phi_t(\alpha) \leq d_1 n^2 (\phi_{MaxRound}(\alpha) + R) + d_2 n \phi_{id}(\alpha) + d_3 n^2.$$

By Proposition 2.4,

$$E_{\phi_t}[H, \Theta] \leq d_1 n^2 E_{\phi_{MaxRound}}[H, \Theta] + d_1 n^2 R + d_2 n E_{\phi_{id}}[H, \Theta] + d_3 n^2.$$

Thus, by Theorem 11.1 and Lemma 12.10, $E_{\phi_t}[H, \Theta] = O(Rn^3)$. ■

Theorem 12.11 gives enough information to derive some time bounds for *DAH*. Here we give some examples. The first item says that whenever all processes are initialized already all non-failing processes decide within expected time $O(Rn^3)$, where R is the number of rounds that are started already. That is, the algorithm has to work for an expected cubic time for each one of the rounds that are started already. The second item says that if we know that at least one of the initialized processes will not fail, then some process decides within expected time $O(Rn^3)$. The third item is an instantiation of the first item saying that all non-failing processes decide within cubic time if at the beginning all processes are initialized and the maximum round number is 1.

Corollary 12.12 *Let H be a fair, well-timed probabilistic execution fragment of DAH that starts from a reachable state s of DAH. The following properties are satisfied by H.*

1. *If in s all processes are initialized already and R is the maximum round of the processes, then within expected time $O(Rn^3)$ all non-failing processes decide.*
2. *If in s there is at least one initialized and non-failed process, the maximum round number is R, and no new process fails, then within expected time $O(Rn^3)$ some process decides.*
3. *If in s all processes are initialized and the maximum round is 1, then within expected time $O(n^3)$ all non-failing processes decide.*

Proof. Item 1 follows from Theorem 12.11 and from the fact that at to reach $\mathcal{D}$ each process must either fail or decide; Item 2 follows from the fact that to reach $\mathcal{D}$ all active processes must decide; Item 3 is an instantiation of Item 1. ■

[[[**NL.** CAN WE SAY SOMETHING ABOUT LARGEST ACTIVE TIME?]]] [[[**RS.** YES, IT IS CUBIC AS WELL. THE PROOF IS QUITE DIFFERENT, THOUGH, UNLESS IT IS POSSIBLE TO IMPROVE WHAT I WROTE LAST SUMMER.]]]

13 Concluding Remarks

[[[**RS.** FIXED 3 TYPOS.]]] We have studied the expected complexity of the randomized consensus algorithm of Aspnes and Herlihy, a highly nontrivial randomized distributed algorithm, and we have developed a collection of math tools that can be used for the analysis of other algorithms as well. Our analysis of the algorithm was driven by two main ideas: decompose the algorithm into simpler parts and separate probability from nondeterminism. The collection of modularization tools that we have developed and their sucessful application show that the analysis of randomized distributed algorithms is indeed feasible and not too difficult. Most of our analysis is essentially the same as the analysis of an ordinary distributed, non-randomized, algorithm.

It is useful to observe the kinds of modularization that we have used and where we have used them. For each kind of modularization we provide a breif description and references to the places in the paper where the modularization results are stated and used, respectively. [[[**RS.** CHANGED THIS PARAGRAPH AND REMOVED FIRST KIND OF MODULARITY.]]]

- Decomposition of a partial progress statement into more statements: progress is achieved through several small easy steps (Proposition 2.8 used in Proposition 8.1).
- Derivation of expected complexity bounds from partial progress statements: an infinitary property is analyzed by means of some finite form of progress (Theorem 2.9 used in Proposition 8.1).
- Modularity of probability spaces with respect to parallel composition (Proposition 2.3 used in Propositions 8.2 and 8.3).
- Coin lemmas and related results to reduce probability to nondeterminism (Theorems 3.5 and 3.7 used in Propositions 9.11 and 9.12 and in Lemma 12.7).
- Transformation of relations between complexity measures into relations between expected complexities. [[[**RS.** REMOVED A SENTENCE.]]] We analyze the complexity of an ordinary execution and we study the relationship between different complexity measures at the level of executions. Then, we transfer the results to probabilistic executions and expected values. (Proposition 2.4 used in Lemmas 12.8 and 12.10 and in Theorem 12.11).
- Analysis of computations divided into phases (Proposition 2.5 used in Lemma 12.10).
- Preservation of expected complexity bounds under parallel composition (Proposition 2.6 used in Lemma 12.9).
- Refinement mappings and related compositionality results (Propositions 2.10, 2.11, and 2.14 used in Theorem 10.2).

[[[**RS.** REORGANIZED THE DISCUSSION BELOW.]]] If we compare the length of our analysis with the length of the original paper of Aspnes and Herlihy, we observe that the two lengths are similar. The length of our analysis is double the length of the analysis in [3]; however, our analysis includes a timing analysis of the protocol, which was not present in [3], and it includes all the details, many of which were not considered in the analysis of [3]. Also, our proof would be considerably shorter if we did not include the detailed invariants and their proofs. These details are not usually included in an algorithm paper.

Even though it is acceptable that the low level details of a proof are omitted in an algorithm paper, we believe that a high level proof should be rigorous enough to avoid the subtleties of randomization, which are due mainly to the interplay between probability and nondeterminism. Intuition often fails when dealing with randomization in a distributed setting. The results that we have presented in this paper provide criteria that allow us not to trip into the subtleties of randomization. We have analyzed a complicated algorithm to make sure that our results are applicable to a wide range of randomized distributed protocols.

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] M. Fischer, N. Lynch, and M. Paterson. Impossibility of distributed consensus with a family of faulty process. *Journal of the ACM*, 32(2):374–382, April 1985.

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

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

[8] L. Lamport. Concurrent reading and writing. *Communications of the ACM*, 20(11):806–811, 1977.

[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$^{\text{th}}$ Annual ACM Symposium on Principles of Programming Languages*, pages 133–138, January 1981.

[10] N.A. Lynch. *Distributed Algorithms.* Morgan Kaufmann Publishers, Inc., 1996.

[11] N.A. Lynch, I. Saias, and R. Segala. Proving time bounds for randomized distributed algorithms. In *Proceedings of the 13th Annual ACM Symposium on Principles of Distributed Computing,* Los Angeles, CA, pages 314–323, 1994.

[12] N.A. Lynch and M.R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing*, pages 137–151, Vancouver, Canada, August 1987. A full version is available as MIT Technical Report MIT/LCS/TR-387.

[13] N.A. Lynch and F.W. Vaandrager. Forward and backward simulations – part I: Untimed systems. Technical Report MIT/LCS/TM-486, MIT Laboratory for Computer Science, May 1993.

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

[15] 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.

[16] 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.

[17] M.Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *Proceedings of 26th IEEE Symposium on Foundations of Computer Science*, pages 327–338, Portland, OR, 1985.