Compositional Verification of Randomized Distributed Algorithms

Roberto Segala

Dipartimento di Scienze dell'Informazione, Università di Bologna
segala@cs.unibo.it

Abstract. We study compositionality issues for the analysis of randomized distributed algorithms. We identify three forms of compositionality that we call process compositionality, property compositionality, and feature compositionality. Process and property compositionality are widely known in the literature, while feature compositionality, although used extensively, does not appear to be emphasized as much. We show how feature compositionality is important for the analysis of randomized systems.

1 Introduction

It is widely recognized that compositionality is an essential feature to enable the scalability of a formal method. That is, it appears to be practically unfeasible to analyze large systems without being able to decompose them into several subcomponents that can be analyzed separately. Compositionality has received considerable attention in the literature; this volume contains several references to the related results. In this paper we study compositionality issues for concurrent systems that contain probability. More specifically, we focus on the analysis of randomized distributed algorithms. We present our study based on the probabilistic model of [38] and on a non-trivial case study [33] where the randomized consensus algorithm of Aspnes and Herlihy [2] is analyzed.

The study of randomization within concurrent systems is particularly complicated due to the interaction of *nondeterminism*, a typical feature of the theory of concurrency, and *probability*, the result of a random choice. The difficulty of randomization is well known in the literature since we can find claims like "intuition often fails to grasp the full intricacy of the algorithm" [31], or "proofs of correctness for probabilistic distributed systems are extremely slippery" [24]. The claims above are further supported by the recent discovery of some problems on known randomized algorithms (e.g., [36, 21, 1]).

In our study of randomized algorithms we have identified three forms of compositionality that we think are important.

– Compositionality of processes.

  This is the typical use of the term "compositionality". That is, we study the properties of a system by subdividing it into several components, studying the properties of each component separately, and then combining the properties of the subcomponents to yield the final result. The properties of the components are sufficiently abstract to hide most of the low level details of the components.

– Compositionality of properties.

  Rather than decomposing a system into several subcomponents, we decompose a property into several simpler properties. Several logics for reasoning about concurrent systems use this form of compositionality.

– Compositionality of features.

  A model for a concurrent system usually includes several paradigms that interfere with each other, e.g., real-time, continuous behavior, probability. The picture is complicated further by the presence of nondeterminism. Rather than studying the system as a whole, we study each paradigm (feature) separately and then combine the results. In this way it is possible to study each feature using its own tools. Being able to separate features simplifies considerably the analysis of an algorithm and reduces the chances of error. In this paper we use *coin lemmas* to separate probability from nondeterminism.

We start by introducing a formal model for the description of randomized distributed algorithms [38]. To enable feature compositionality the model is an extension of *Labeled Transition Systems* (LTSs) [20], since there is an extensive literature on the analysis of LTSs that can be adapted to the probabilistic case. In our extension of LTSs we use the synchronization mechanism of CSP [18], where processes synchronize on common actions and evolve independently on others. Our choice of the CSP synchronization style derives from the fact that it allows us to model easily distributed algorithms. We call our probabilistic LTSs *Probabilistic Automata*. Some important properties of probabilistic automata are the following.

– An ordinary LTS is a special case of a probabilistic automaton.
– The main properties that enable compositional reasoning, e.g., projections of executions, are preserved.

Once probabilistic automata are defined, we introduce a generic notion of a *complexity measure* and the related notion of *expected complexity* of an algorithm. We show how it is possible to lift a property of complexity measures to a property of expected complexities as an example of feature compositionality.

We introduce *progress statements* [25, 38], a probabilistic generalization of the *leadsto* operator of UNITY [6], to illustrate how a complex property can be decomposed into simpler properties (property compositionality) and to illustrate a technique to derive expected complexity bounds for an algorithm (feature compositionality).

Finally, we introduce *coin lemmas* [25, 38] to illustrate our main technique to separate probability from nondeterminism. Coin lemmas are a formal expression of the intuition that a randomized algorithm behaves correctly whenever some specific random draws give some specific results. Coin lemmas provide us with a technique to reduce the analysis of a randomized system to the analysis of an ordinary LTS.

As our main running example we use a large case study [33] where the randomized consensus algorithm of Aspnes and Herlihy [2] is shown to terminate within polynomial time. The algorithm of Aspnes and Herlihy is particularly interesting for its high non-triviality. Its nondeterministic behavior is very complicate, and its probabilistic behavior is based on the theory of Random Walks [13]. In this case study all the forms of compositionality that we introduce are used.

The rest of the paper is organized as follows. Section 2 introduces the model [38] that we use for the analysis of randomized distributed algorithms; Section 3 introduces complexity measures [38, 32] and shows how to study the expected complexity of an algorithm; Section 4 introduces progress statements [38, 25, 32] for the study of the partial progress of an algorithm; Section 5 describes the algorithm of Aspnes and Herlihy, our running example; Section 6 shows how progress statements and projections can be used to reason compositionally about an algorithm; Section 7 describes the main probabilistic component of the algorithm of Aspnes and Herlihy, and Section 8 shows how to use coin lemmas to reason compositionally about the probabilistic behavior of an algorithm; Section 9 gives a hint on how to use refinements [27] to reason about randomized algorithms; Section 10 analyzes the time complexity of the algorithm of Aspnes and Herlihy and gives examples of how to reason compositionally using complexity measures; finally, Section 11 gives references to related work, and Section 12 gives some concluding remarks.

2 Probabilistic Automata

In this section we introduce probabilistic automata by enriching the probabilistic automata of [38] with an input/output distinction. The input/output distinction is useful to define some meaningful fairness conditions; however, the properties that we describe are valid even without such distinction.

2.1 Probability Spaces

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

A probability space $(\Omega, \mathcal{F}, P)$ is *discrete* if $\mathcal{F} = 2^{\Omega}$ and for each $C \subseteq \Omega$, $P[C] = \sum_{x \in C} P[\{x\}]$. For any arbitrary set X, let $Probs(X)$ denote the set of discrete probability spaces $(\Omega, \mathcal{F}, P)$ where $\Omega \subseteq X$, and such that all the elements of Ω have a non-zero probability.

2.2 Probabilistic 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 of input, output, and internal actions, respectively;
- a transition relation $Trans(A) \subseteq States(A) \times Actions(A) \times States(A)$, where $Actions(A)$ denotes the set $in(A) \cup out(A) \cup int(A)$, such that for each state s of $States(A)$ and each input action a of $in(A)$ there is a state s' such that $(s, a, s') \in Trans(A)$;
- a task partition $Tasks(A)$, which is an equivalence relation on $int(A) \cup out(A)$ that has at most countably many equivalence classes. The elements of $Trans(A)$ are called *transitions*, and A is said to be *input enabled*. An equivalence class of $Tasks(A)$ is called a *task* of A.

A *probabilistic I/O automaton* M differs from an I/O automaton in its transition relation. That is, $Trans(M) \subseteq States(M) \times Actions(M) \times Probs(States(M))$. In the rest of the paper we refer to (probabilistic) I/O automata as (probabilistic) automata. Observe that an automaton is a special case of a probabilistic automaton.

Probabilistic automata are partially captured by the reactive model of [16] in the sense that the reactive model assumes some form of nondeterminism between different actions. However, the reactive model does not allow nondeterministic choices between transitions involving the same action. By restricting simple probabilistic automata to have finitely many states, we obtain objects with a structure similar to that of the Concurrent Labeled Markov Chains of [17]; however, in our model we do not need to distinguish between nondeterministic and probabilistic states. In our model nondeterminism is obtained by means of the structure of the transition relation. This allows us to retain most of the traditional notation that is used for automata.

2.3 Executions

A state s of M is said to *enable* a transition if there is a transition $(s, a, \mathcal{P})$ in $Trans(M)$. An action a is said to be enabled from a state s of M if s enables a transition with action a.

An *execution fragment* of M is a sequence α of alternating states and actions of M starting with a state, and, if α is finite ending with a state, $\alpha = s_0 a_1 s_1 a_2 s_2 ...$, such that for each $i \geq 0$ there exists a transition $(s_i, a_{i+1}, \mathcal{P})$ of M such that $s_{i+1} \in \Omega$. Denote by $fstate(\alpha)$ the first state of α and, if α is finite, denote by $lstate(\alpha)$ the last state of α. An *execution* is an execution fragment whose first state is a start state.

An execution fragment α is said to be *fair* iff the following conditions hold for every task T of M:

1. if α is finite then no action from T is enabled in $lstate(\alpha)$;
2. if α is infinite, then either actions from T occur infinitely many times in α, or α contains infinitely many occurrences of states from which no action from T is enabled.

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

2.4 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. It has a structure similar to the structure of a probabilistic automaton, where the states are finite execution fragments of M.

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 of transitions of M $\{(lstate(q), a_i, \mathcal{P}_i)\}_{i \geq 0}$ and a family of probabilities $\{p_i\}_{i \geq 0}$ 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, $P[(a, qas)] = \sum_{i | a_i = a} p_i P_i[s]$.

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 $1 - p$.

It is possible to define a probability space $\mathcal{P}_H = (\Omega_H, \mathcal{F}_H, P_H)$ associated with H. In particular Ω_H is a set of execution fragments of M (the limit closure of the states of H), $\mathcal{F}_H$ is the smallest σ-field that contains the set of *cones* C_q, consisting of those elements of Ω_H having q as a prefix (let q denote a state of H), and the probability measure P_H is the unique extension of the probability measure defined on cones as follows: $P_H[C_q]$ is the product of the probabilities of each transition of H leading to q. Standard measure theory guarantees that $\mathcal{P}_H$ is well defined. Furthermore, $\mathcal{P}_H$ is sufficiently rich to describe properties like single or multiple occurrences of an action, reachability properties, fairness properties. See [38] for more details.

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. We have chosen the term finitely satisfiable since it is possible to determine that an execution α is in E by looking at a finite prefix of α. A finitely satisfiable event can be represented by a set Θ of incomparable states of H. The event denoted by Θ is $\cup_{q \in \Theta} C_q$. We abuse notation by writing $P_H[\Theta]$ for $P_H[\cup_{q \in \Theta} C_q]$. We call a set of incomparable states of H a *cut* of H, and we say that a cut Θ is *full* if $P_H[\Theta] = 1$.

An important event of $\mathcal{P}_H$ is the set of fair executions of Ω_H. We define a probabilistic execution fragment H to be fair if the set of fair execution fragments has probability 1 in $\mathcal{P}_H$.

2.5 Parallel Composition

Probabilistic automata can be composed in parallel. Due to the reactive structure of probabilistic automata, the definition of parallel composition is simple. The states of the composition are the cross product of the states of the components. The composed probabilistic automata synchronize on their common actions and evolve independently on the others. Whenever a synchronization occurs, the state that is reached is obtained by choosing a state independently for each of the probabilistic automata involved.

Formally, 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 $M_1 \| M_2$, 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. $in(M) = (in(M_1) \cup in(M_2)) - (out(M_1) \cup out(M_2)$,
   $int(M) = int(M_1) \cup int(M_2)$,
   $out(M) = out(M_1) \cup out(M_2)$,
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.

In a parallel composition the notion of *projection* is one of the main tools to support compositional reasoning. A projection of an execution fragment α onto a component within a parallel composition is the contribution of the component to obtain α. Formally, let M be $M_1 \parallel M_2$, the parallel composition of M_1 and M_2, and let α be an execution fragment of M. The projection of α onto M_i, denoted by $\alpha\lceil M_i$, is the sequence obtained from α by replacing each state with its i^{th} component and by removing all actions that are not actions of M_i together with their following state. It is the case that $\alpha\lceil M_i$ is an execution fragment of M_i [26].

The notion of projection can be extended to probabilistic executions (cf. Section 4.3 of [38]). Here we do not present the formal definition of projection; rather, we describe the properties of projections that are needed for our analysis, and we refer the reader to [38] 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* (cf. Proposition 1) of the probability space associated with H. This property allows us to prove probabilistic properties of H based on probabilistic properties of $H\lceil M_i$ (process compositionality).

Proposition 1 [33]. *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 $\Theta \in \mathcal{F}_{H\lceil M_i}$, $P_{H\lceil M_i}[\Theta] = P_H[\{\alpha \in \Omega_H \mid \alpha\lceil M_i \in \Theta\}]$.* □

3 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, where by complexity of an execution fragment we mean something proportional to the amount of work that is neessary to carry out the related operations. 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. Examples of complexity measures are the total number of operations performed in a protocol, and the number of operations of some specific type performed in a protocol.

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

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

The expected complexity $E_\phi[H, \Theta]$ expresses the average amount of work necessary before satisfying the property expressed by Θ. If the probability of Θ is not 1, then the average work could be potentially infinite.

Below we present three compositionality results for complexity measures. Proposition 2 is an instance of feture compositionality, Proposition 3 is an instance of property compositionality, and Proposition 4 is an instance of process compositionality. We give an informal explanation of each result.

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. This result follows from the observation that the function that expresses the complexity of the elements of a full cut is a random variable [13].

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

Suppose that within a computation it is possible to identify several phases, each one with its own complexity, and suppose that the complexity associated with each phase remains 0 until the phase starts. Suppose that the expected complexity of each phase is bounded by some constant c. If we know that the expected number of phases that start is bounded by k, then the expected complexity of the system is bounded by ck. In the statement below ϕ_i denotes the complexity associated with phase i and ϕ denotes the number of phases that have started.

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

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

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

Finally, to verify properties modularly it is useful to derive complexity properties of complex systems based on complexity properties of their components. Proposition 4 below is an example of how to lift an expected complexity bound from a component to the whole composition.

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

4 Progress Statements

A progress statement is a predicate that can be used to state reachability properties. It is a probabilistic extension of the leadsto operator of UNITY [6]. The notation for a progress statement is

$$U \xrightarrow[p]{\phi \leq c} U',$$

where U and U' are sets of states, p is a probability, ϕ is a complexity measure, and c is non-negative real number. It states that, no matter how the nondeterminism is resolved, the probability of reaching a state of U' from a state of U within ϕ-complexity c is at least p. In this paper we require fairness for the resolution of nondeterminism; however, the results that we describe below hold also for more general schemas of resolution of the nondeterminism [38].

Given a probabilistic execution fragment H of a probabilistic automaton M, let $e_{U',\phi(c)}(H)$ denote the set of executions α of Ω_H with a prefix α' such that $\phi(\alpha') \leq c$ and $lstate(\alpha') \in U'$. We say that the predicate $U \xrightarrow[p]{\phi \leq c} U'$ is true for M iff for each fair probabilistic execution fragment H of M that starts from a state of U, $P_H[e_{U',\phi(c)}(H)] \geq p$.

Progress statements can be decomposed into simpler statements to be proved separately. Some examples of decompositions are provided by the proposition below.

Proposition 5. *Let M be a probabilistic automaton, $U, U', U'', U''' \subseteq States(M)$, and 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 $U \cup U'' \xrightarrow[p]{\phi \leq c} U' \cup U''$;*
3. *if $U \xrightarrow[p]{\phi \leq c} U'$ and $U'' \xrightarrow[p']{\phi \leq c'} U'''$, then $U \cup U'' \xrightarrow[min(p,p')]{\phi \leq max(c,c')} U' \cup U'''$.* □

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

Proposition 6 [38]. *Let M be a probabilistic automaton and ϕ be a complexity measure for M. Suppose that for each execution fragment of M of the form sas', $\phi(sas') \leq 1$, that is, each transition of M increases ϕ by at most 1. Let U and U' be sets of states of M. Let H be a probabilistic execution fragment of M that starts from a state of U, and suppose that for each state q of H such that $lstate(q) \in U - U'$ some transition is scheduled with probability 1. Suppose also that $U \xrightarrow[p]{\phi \leq c} U'$ and $U \Rightarrow U \, unless \, U'$. Then, $E_\phi[H, \Theta_{U'}(H)] \leq (c+1)/p$.* □

5 Example: The Algorithm of Aspnes and Herlihy

The algorithm of Aspnes and Herlihy [2] is a randomized algorithm that solves the *consensus* problem within expected polynomial time. The problem consists of letting n processes agree on some value in the set $\{0, 1\}$ so that the properties of *validity*, *agreement*, and *wait-free termination* are satisfied. Validity states that the value chosen by each process should be a value that was proposed in the past by some process; agreement states that no two processes choose different values; wait-free termination states that all non-failed processes eventually decide. Processes may fail by stopping, and the interaction between processes is asynchronous. In other words, it is not possible to distinguish between a slow process and a failed process. It is shown in [14] that there is no algorithm that can solve the consensus problem. Aspnes and Herlihy have shown that by relaxing the wait-free termination property so that the probability of termination is 1 the consensus problem can be solved within expected polynomial time.

The algorithm of Aspnes and Herlihy proceeds in rounds. Every process maintains a variable with two fields, *value* and *round*, that contain the process' current preferred value ($0, 1$ or $\perp$) and current round (a non-negative integer), respectively. We say that a process is at round r if its *round* field is equal to r. The variables $(value, round)$ are multiple-reader single-writer. Each process starts with its *round* field initialized to 0 and its *value* field initialized to $\perp$.

After receiving the initial value to agree on, each process i executes the following loop. It first reads the $(value, round)$ variables of all other processes in its local memory. We say that process i is a *leader* if according to its readings its own round is greater than or equal to the rounds of all other processes. We also say that a process i *observed* that another process j is a leader if according to i's readings the round of j is greater than or equal to the rounds of all other processes. If process i at round r discovers that it is a leader, and that 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 again the other processes. If once again the leaders observed by i do not agree, then i determines its new preferred value for the next round by invoking a coin flipping protocol. There is a separate coin flipping protocol for each round.

We represent the main part of the algorithm as an automaton AP (Agreement Protocol) and the coin flipping protocols as probabilistic automata CF_r (Coin Flipper), one for each round r (cf. Figure 1). The coin flipper receives and handles

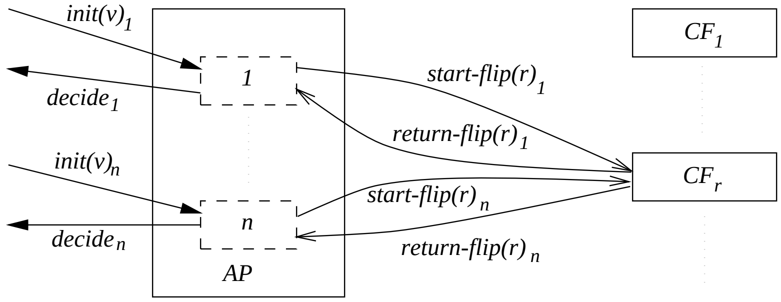

Fig. 1. The decomposition of the algorithm of Aspnes and Herlihy.

requests from each process. We say that a request from a process i is received on *port* i, and we say that a port i is non-failing if process i does not fail. With this decomposition we can analyze several properties just on AP using ordinary techniques for non-probabilistic systems. Indeed, in this section we deal with AP only, and we leave the coin flippers unspecified.

The formal definition of AP is given in Table 1 using the precondition/effect notation that is typical of I/O automata [26]. Beside the shared variables $value(i)$ and $round(i)$, each process has a program counter pc, two arrays $values$ and $rounds$ containing the scans of the other processes, a set variable obs saying what processes have been observed, a variable $start$ holding the initial preferred value, and two variables $decided$, and $stopped$ stating whether the process has decided or failed. We explain some of the relevant predicates: $obs\text{-}leader(j)$ 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 i observes that the leaders agree on a value v; $obs\text{-}leader\text{-}value$ is the value of one of the leaders observed by i. We say that a process is *active* if it is attempting to agree on a value. An active process becomes inactive either by deciding a value or by failing.

6 Compositionality Using Projections and Progress Statements

The validity and agreement properties of the algorithm of Aspnes and Herlihy do not depend on any probabilistic assumption. These are *safety* properties and can be studied solely on AP by means of ordinary invariants. Informally, the invariant for validity states that no process will ever prefer a value different from its initial value if all processes have the same initial value, while the invariant for agreement states that if a process i that is at round r is "about to decide" on some value v, then every

Actions and transitions of process i.

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

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

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

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

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

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

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

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

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

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

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

Table 1. The actions and transition relation of AP.

process that is at round r or higher has its value equal to v. Since the verification of validity and agreement is reduced to the analysis of ordinary nondeterministic systems, it is not our interest here to pursue such direction and we refer the reader to [33] for further details.

For the wait-free termination property we prove that the algorithms terminates within an expected constant number of rounds and we relate later the round complexity to the time complexity. That is, we prove the following.

Theorem 7. *The algorithm of Aspnes and Herlihy terminates within a constant expected number of rounds.* $\square$

We use progress statements to derive our result. Define the following sets of states.

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

Let $\phi_{MaxRound}$ be a complexity measure that counts the number of new rounds visited within an execution fragment, i.e., $\phi_{MaxRound}(\alpha) = \mathit{lstate}(\alpha).\mathit{max\text{-}round} - \mathit{fstate}(\alpha).\mathit{rmax}$, where $s.\mathit{max\text{-}round}$ denotes the highest round number of the processes in state s. Our objective is to show that the progress statement

$$\mathcal{R} \xrightarrow[p]{\phi_{MaxRound} \leq 3} \mathcal{D} \tag{1}$$

is valid for a number p that is independent of n, the number of precesses. Then, using Proposition 6, we can derive from (1) that a state of $\mathcal{D}$ is reached within expected $4/p$ rounds, that is, within a constant expected number of rounds. Note that fairness implies that from every state of $\mathcal{R} - \mathcal{D}$ the probability of scheduling a transition is 1, thus satisfying the condition for the applicability of Proposition 6.

The advantage of using the progress statement (1) is that we are left with a property that can be verified by analyzing a finite number of rounds. However, the analysis of Statement (1) is still too complex. The informal argument to prove Statement (1) argues that either a decision is reached, or eventually some process moves to a new fresh round. Once a new round is reached, we know that no coin has been flipped yet at that round. Furthermore, if all the coins flipped at the new round give the same result, then a decision is reached within two other rounds. In order to reflect the informal argument, we decompose Statement (1) into two parts (property compositionality) and use Proposition 5 to combine them. For $v \in \{0, 1\}$, define the following set of states.

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

Then,

$$\mathcal{R} \xrightarrow[1]{\phi_{MaxRound} \leq 1} \mathcal{F}_0 \cup \mathcal{F}_1 \cup \mathcal{D} \tag{2}$$

and

$$\mathcal{F}_v \xrightarrow[p]{\phi_{MaxRound} \leq 2} \mathcal{D}. \tag{3}$$

In order to show the validity of Statement (2) and (3) we identify two properties of AP and two properties of CF that can be composed together to yield the final result (process compositionality). The properties of AP are the following:

D1 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.
D2 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.

The properties of each CF_r are the following.

C1 For each fair probabilistic execution fragment of CF_r that starts with a reachable state of CF_r, the probability that each invocation on a non-failing port gets a response is 1.
C2 For each fair probabilistic execution of CF_r, and each value $v \in \{0, 1\}$, the probability that all invocations on a non-failing port get response v is at least p, $0 < p \leq 1$.

Properties **D1** and **D2** are properties of an ordinary nondeterministic system and can be analyzed by means of existing techniques (invariants and liveness arguments). Therefore, we do not deal with their analysis in this paper. Properties **C1** and **C2** are properties of the coin flippers. Their proofs involve the analysis of a random walk [13] and are postponed to the following sections. We emphasize that, by decomposing the system into AP and CF, the main probabilistic analysis of the algorithm is done on the coin flippers only.

Our final objective for this section is to show how properties **D1**, **D2**, **C1**, and **C2** can be composed to yield Statements (2) and (3). To this purpose we use the results about the projections of a probabilistic execution (cf. Proposition 1).

Proposition 8. *Assuming that properties* **C1** *and* **D1** *are valid, Statement (2) is valid.*

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 the definition of projection, the executions of $\Theta\lceil AP$ satisfy the premise of **D1**, and thus 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 1, $P_H[\Theta_i] = 1$. Therefore, $P_H[\Theta] = 1$ since, from probability theory, any countable intersection of probability 1 events has probability 1.

Proposition 9. *Assuming that properties* **D1**, **D2**, **C1** *and* **C2** *are valid, Statement (3) is valid.*

Proof. Let H be a probabilistic execution fragment of AH that starts from a state s_0 of $\mathcal{F}_v$, and let $r = s_0.max\text{-}round$. 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 the definition of projection, the executions of $\Theta\lceil AP$ satisfy the premise of **D2**, and thus, by **D2**, 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, $\Theta = (\cap_{i\geq 1}\Theta_i) \cap \Theta'_r$. 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, and Θ'_r is the inverse image under projection of the set of executions of $\Omega_{H\lceil CF_r}$ where each response has value v. From **C1**, for each i, $P_{H\lceil CF_i}[\Theta_i\lceil CF_i] = 1$, and thus, by Proposition 1, $P_H[\Theta_i] = 1$. Since $s_0 \in \mathcal{F}_v$ and $r = s_0.max\text{-}round$, $H\lceil CF_r$ 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 1, $P_H[\Theta'_r] \geq p$. 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.

7 Example: a Coin Flipping Protocol

The algorithm of Aspnes and Herlihy relies on a coin flipper that satisfies the properties **C1** and **C2** mentioned earlier. The algorithm for the coin flipper is given in terms of n processes that interact through a centralized multiple-write single-read counter. Each of the n processes works as follows: once a request for a flip is received, it reads the value of the counter to check whether it is beyond one of the barriers Kn or $-Kn$, where K is a fixed constant. If the counter is above Kn, then the process returns value 1; if the counter is below $-Kn$, then the process returns 0; otherwise, the process first flips a fair coin to decide whether to increment or decrement the value of the counter, and then starts again.

If after each coin flip we look at the difference between the heads and tails obtained so far, we observe that this number increases/decreases with probability 1/2 at each step. This process is called a *random walk*. By looking at the structure of the coin flipping protocol, we observe that the value of the shared counter may differ by at most n from the current value of the difference between heads and tails.

Therefore, as soon as the difference between heads and tails goes beyond one of the barriers $\pm(K+1)n$, all the processes return a value. From random walk theory this property holds with probability 1. Furthermore, if the barrier $(K+1)n$ is reached before the barrier $-(K-1)n$, then all the processes return 1. From random walk theory this property holds with probability $(K-1)/2K$, our p in property **C2**. A symmetric argument holds for the case where all processes return 0.

So far we have argued informally that the coin flipping protocol works correctly since it behaves like a random walk. However, how can we be sure that there is really a random walk going on in the algorithm? Is there any way that the nondeterminism can affect the randomized process we have identified? For example, how can we guarantee that the scheduler cannot prevent a process from flipping whenever one of the barriers $\pm Kn$ is too close? Indeed, the scheduler can prevent processes from flipping; fortunately, this situation occurs only if all the processes that are flipping fail. The main problem here is that the above argument did not appear in our informal analysis, and the absence of such an argument could be the source of errors in general (cf. [36] for an example).

Our approach to this problem is to provide general theorems that separate the probabilistic argument (properties of the random walk) from the nondeterministic argument (how the scheduler can affect the random walk), so that each problem can be analyzed in its own field. We call such results *coin lemmas* [38, 25, 32], which are an instance of feature compositionality.

8 Compositionality Using Coin Lemmas

A useful technique to prove the validity of a probabilistic property for a probabilistic automaton M is the following [32]:

1. choose a set of random draws that may occur within a probabilistic execution of M, and choose some of the possible outcomes;
2. show that, no matter how the nondeterminism is resolved, the chosen random draws give the chosen outcomes with some minimum probability p;
3. show that whenever the chosen random draws give the chosen outcome, a state from U' is reached within c units of complexity ϕ.

The first two steps can be carried out using the so-called *coin lemmas* [25, 32, 38], 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. We refer the reader to [38] for several examples of coin lemmas. Here we illustrate a coin lemma for symmetric random walks.

8.1 A Coin Lemma for Random Walks

Roughly speaking, a random walk is a process that describes the moves of a particle on the real line, where at each time the particle moves in one direction with probability p and in the opposite direction with probability $(1-p)$. In this section we present a coin lemma for symmetric random walks. That is, we present a rule for choosing events within a probabilistic execution fragment that are guaranteed to have properties similar to the properties of random walks where $p = 1/2$. A more general result is given in [33].

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

Proposition 10. *Let H be a probabilistic execution fragment of M, and let $B \leq z \leq T$. Then*

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

Therefore, we know lower bounds on the probability of the events expressed by **Top**, **Bot**, and **Either**, which are closely connected with our informal argument of correctness for the coin flipping protocol. Note that the events contain executions where finitely many coins are flipped and no barrier is reached. During the analysis of the events (with no probability involved) these executions appear, and therefore we are forced to analyze the case where the scheduler prevents the protocol from reaching one of the barriers.

We conclude with a result about the expected complexity of a random walk. Let $\phi_{Acts}(\alpha)$ be the complexity measure that counts the number of actions from $Acts$

that occur in α. Define $\phi_{Acts,B,T,z}$ to be the truncation of ϕ_{Acts} at the point where one of the barriers B and T is reached. Then we can prove an upper bound on the number of expected flip actions that occur before reaching one of the barriers.

Proposition 11. *Let H be a probabilistic execution fragment of M, and let Θ be a full cut of H. Let $B \leq z \leq T$. Then, $E_{\phi_{Acts,B,T,z}}[H, \Theta] \leq -z^2 + (B+T)z - BT$.* $\square$

8.2 Analysis of the Coin Flipping Protocol

We build a coin flipping protocol that satisfies **C1** and **C2** with $p = (K-1)/2K$. The protocol is based on random walks. We define the protocol by letting a probabilistic automaton DCN_r (Distributed CoiN) interact with a non-probabilistic counter CT_r (CounTer), that is, $CF_r = DCN_r \parallel CT_r$. In this Section, DCN_r is distributed while CT_r is composed of n processes that receive requests from DCN_r and read/update a single shared variable. In Section 9 we discuss how to decentralize CT_r. Since the protocols for DCN_r and CT_r are the same for any round r, we drop the subscript r from our notation. In DCN each process flips a fair coin to decide whether to increment or decrement the shared counter. Then the process reads the current value of the shared counter by invoking CT, and if the value read is beyond the barrier $-Kn$ ($+Kn$), where K is a fixed constant, then the process returns 0 (1). The specification of CT states that an increment or decrement operation always completes unless the corresponding process fails, while a read operation is guaranteed to complete only if increments and decrements eventually cease.

For the analysis of the coin flipping protocol we start with part 3. Let $Acts$ be $\{flip_1, \ldots, flip_n\}$, and let $\mathbf{S}$ be $\{(U_1^i, U_1^d), (U_2^i, U_2^d), \ldots, (U_n^i, U_n^d)\}$, where U_j^i is the set of states of CF where process j has just flipped inc ($fpc_j = inc$), and U_j^d is the set of states of CF where process j has just flipped dec ($fpc_j = dec$). Given a finite execution fragment α of CF, let $\phi_{inc}(\alpha)$ be the number of coin flips in α that give inc, and let $\phi_{dec}(\alpha)$ be the number of coin flips in α that give dec.

Lemma 12. *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 in α each invocation on a non-failing port gets a response.* $\square$

Lemma 13. *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 in α every invocation on a non-failing port gets response 1.* $\square$

The proofs of Lemmas 12 and 13 follow from simple invariant properties and do not involve probability. The main idea is that the value of the shared counter remains beyond Kn ($-Kn$) once the barrier $(K+1)n$ ($-(K+1)n$) is reached. A symmetric argument is valid for **Bottom**$[-(K-1)n, (K+1)n, 0](H)$.

At this point properties **C1** and **C2** can be proved by simple applications of the coin lemma for random walks.

Proposition 14. *The coin flipper CF satisfies* **C1**. *That is, for each fair probabilistic execution fragment of CF that starts with a reachable state of CF, with probability 1 each invocation on a non-failing port gets an answer.*

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 12, every invocation to CF in $\alpha \frown \alpha'$ gets a response, and therefore every invocation to CF in α' gets a response. By Theorem 10, $P_H[\mathbf{Either}[-(K+1)n, (K+1)n, z](H)] = 1$.

Proposition 15. *The coin flipper CF satisfies* **C2** *with* $p = (K+1)/2K$. *That is, fixed* $v \in \{0, 1\}$*, for each fair probabilistic execution of CF, with probability at least* $(K-1)/2K$ *each invocation to CF on a non-failing port returns value* v.

Proof. Assume that $v = 1$; the case for $v = 0$ is symmetric. Let H be a fair probabilistic execution of CF. If α is an execution of $\mathbf{Top}[-(K-1)n, (K+1)n, 0](H)$, then, by Lemma 13, every invocation to CF in α gets response 1. By Theorem 10, $P_H[\mathbf{Top}[-(K-1)n, (K+1)n, 0](H)] \geq (K-1)/2K$.

9 Compositionality Using Refinements

A well known compositional verification technique for ordinary automata is based on the notions of *forward/backward simulation* [27] and *bisimulation* [29]. These notions can be extended to probabilistic automata as well [22, 39]. The simulation method is sound for the notion of *trace inclusion* [27], which can also be used as a notion of implementation. The same is true for probabilistic automata [37, 38], and the algorithm of Aspnes and Herlihy provides again a significative example of how probabilistic simulation relations enable compositional reasoning.

In order for the algorithm of Aspnes and Herlihy to be really wait-free, the counter CT must be distributed among all the processes of a system. The distributed implementation of CT, which we denote by DCT (Distributed CounTer), is presented in [2]. We are not interested in the details of the implementation here since there is no probability involved. It is possible to verify that DCT implements CT by exhibiting a *refinement mapping* [27] from DCT to CT. This part of the proof is simple and does not involve probability. Then we use the fact that an ordinary refinement is a special case of a probabilistic refinement, and the fact that the existence of refinements is preserved by parallel composition to lift to the whole algorithm the refinement from DCT to CT, thus showing that DCT can replace CT in AH.

We emphasize that in the analysis above there is no probability involved. The decomposition of the coin flipping protocol into two parts, the processes that do flip coins and the shared counter, allows us to use probabilistic arguments only in those places where probability is really involved.

10 Time Analysis by Lifting Complexity Measures

In this section we derive an upper bound on the time to reach $\mathcal{D}$ once all processes have some minimum speed. We achieve this result by studying the expected number of *inc* and *dec* events (increments and decrements of the shared counter) that occur within the coin flippers and then converting the new expected bound into a time bound. This is done by studying several properties that express relationships between different complexity measures, and then lifting the results to expectations, again an example of feature compositionality. Again we omit all the details that do not rely on probability and we refer the interested reader to [33].

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

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

We start with some non-probabilistic properties about the new complexity measures. The first result, Lemma 16, provides a linear upper bound on the time it takes for DAH to span a given number of rounds and to flip a given number of coins under the assumption of well-timedness. The next two results state basic properties of the coin flipping protocols. That is, once a barrier $\pm(K+1)n$ is reached, there are at most n other *flip* events, and within any execution fragment of DCF_r the difference between the *inc*, *dec* events and the *flip* events is at most n.

Lemma 16. *Let α be a well-timed execution fragment of DAH, and suppose that all the states of α, with the possible exception of $lstate(\alpha)$ are active, that is, are states of $\mathcal{R}$. Let $R = fstate(\alpha).max\text{-}round$. Then, $\phi_t(\alpha) \leq d_1 n^2(\phi_{MaxRound}(\alpha) + R) + d_2 n \phi_{id}(\alpha) + d_3 n^2$ for some constants d_1, d_2, and d_3.* □

Lemma 17. *Let $\alpha = \alpha_1 \frown \alpha_2$ be a finite execution of DCF_r, and suppose that $|Diff_{Acts,\mathbf{S}}(\alpha_1)| \geq (K+1)n$. Then $\phi_{flip,r}(\alpha_2) \leq n$.* □

Lemma 18. *Let α be a finite execution fragment of DCF_r. Then, $\phi_{id,r}(\alpha) \leq \phi_{flip,r}(\alpha) + n$.* □

We now deal with probabilistic properties. First, based on our results on random walks and on Lemma 17, we show in Lemma 19 an upper bound on the expected number of coin flips performed by a coin flipper. Then, in Lemma 20 we use Lemma 18 and our results about linear combinations of complexity measures to derive an upper bound on the expected number of increment and decrement operations performed by a coin flipper, and we use our compositionality result about complexity measures to show that the bound is preserved by parallel composition. Finally, in Lemma 21 we use our result about phases of computations to combine Theorem 7 with Lemma 20 and derive an upper bound on the expected number of *inc* and *dec* events performed by the algorithm.

Lemma 19. *Let H be a probabilistic execution fragment of DCF_r that starts from a reachable state, and let Θ be a full cut of H. Then $E_{\phi_{flip,r}}[H,\Theta] \leq (K+1)^2 n^2 + n$.* □

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 17, 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 11, $E_{\phi_{Acts,-(K+1)n,(K+1)n,z}}[H,\Theta] \leq -z^2 + (K+1)^2 n^2 \leq (K+1)^2 n^2$, 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 17 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, $E_{\phi_{flip,r}}[H,\Theta] \leq (K+1)^2 n^2 + n$.

Lemma 20. *Let H be a probabilistic execution fragment of DAH that starts from a reachable state, and let Θ be a full cut of H. Then $E_{\phi_{id,r}}[H,\Theta] \leq (K+1)^2 n^2 + 2n$.* □

Proof. By Lemma 18, for each execution fragment of α of CF_r, $\phi_{id,r}(\alpha) \leq \phi_{flip,r}(\alpha) + n$. By Proposition 2, $E_{\phi_{id,r}}[H,\Theta] \leq E_{\phi_{flip,r}}[H,\Theta] + n$. By Lemma 19, $E_{\phi_{flip,r}}[H,\Theta] \leq (K+1)^2 n^2 + n$. Thus, $E_{\phi_{id,r}}[H,\Theta] \leq (K+1)^2 n^2 + 2n$.

Lemma 21. *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 20 and Proposition 4, there is a constant $c = (K+1)^2n^2 + 2n$ 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 Proposition 3: 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 3, $E_{\phi'}[H, \Theta] \leq cE_{\phi_{MaxRound}}[H, \Theta]$. By Theorem 7, $E_{\phi_{MaxRound}}[H, \Theta]$ 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, $E_{\phi_{id}}[H, \Theta] = O(Rn^2) + O(n^2) = O(Rn^2)$.

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

Theorem 22. *Let H be a probabilistic fair, well-timed execution fragment of DAH with a reachable start state s, and let $R = s.max\text{-}round$. Let Θ denote the set of minimal states of H where a state from $\mathcal{D}$ is reached. Then $E_{\phi_t}[H, \Theta] = O(Rn^3)$.*

Proof. By Lemma 16 and Proposition 2, $E_{\phi_t}[H, \Theta] \leq d_1 n^2 E_{\phi_{MaxRound}}[H, \Theta] + d_1 n^2 R + d_2 n E_{\phi_{id}}[H, \Theta] + d_3 n^2$. Thus, by Theorem 7 and Lemma 21, $E_{\phi_t}[H, \Theta] = O(Rn^3)$. □

11 Related Work

There is an extensive literature on the description and analysis of randomized systems. Objects with the same structure as probabilistic automata were introduced already by Rabin [34], even though with different motivations and objectives.

From the modeling point of view there are several results in process algebras [23, 15, 42, 3, 44, 8, 7, 9, 45, 10, 46, 17, 40], where algebras like CCS [28], CSP [18], and ACP [5] are enriched with probability. Most of the algebras above do not deal with nondeterminism and can be classified into *reactive*, *generative*, and *stratified* according to [16]. Our probabilistic automata are an extension of the reactive model of [16], while our probabilistic executions are an example of a generative process. The algebras of [17, 45] do include nondeterminism. The algebra of [45] is used to study a theory of testing for probabilistic systems; the algebra of [17] is based on the alternating model of [43] and is used mainly to illustrate a new model checking algorithm for probabilistic systems. In the alternating model there is a strict alternation between states that enable only nondeterministic transitions and states that enable a single probabilistic transitions. In our model we avoid the alternation, thus obtaining a structure which is closer to ordinary automata.

Other techniques for the analysis of randomized algorithms are studied in [30, 31, 35, 47, 12, 41]. Most of the work concentrates on properties that hold with probability 1 and that can be verified simply by looking at the topology of a system. In [30] a notion of extreme fairness is introduced, later generalized in [47] under the name of α-fairness. The set of α-fair executions of a system have probability 1; thus, the correctness of a system can be verified just by looking at its α-fair executions. The problem with the study of properties that hold with probability 1 is that it is not easy to study the expected complexity of a system.

In [41] a reasoning in the style of weak preconditions is extended to probabilistic systems using objects called predicate transformers. The method, though, seems to be applicable only to small systems. It would be useful to investigate how methods like those of [41] can be integrated with our way of reasoning about systems. In [12] a different approach to the analysis of a randomized algorithm is presented. An algorithm is viewed as a game between a player called *scheduler*, which tries to degrade performance, and a player called *luck*, which fixes the outcome of some coins trying to improve performance. We say that luck has a winning strategy with k moves if luck can make the algorithm work against any scheduler by fixing the value of at most k coins. In such case it is possible to show that the algorithm works with probability at least $1/2^k$. This approach can be seen as an instance of coin lemmas, where the game is the rule to map the process of flipping k coins onto a probabilistic execution.

12 Concluding Remarks

We have shown how different forms of compositionality can be included into a model for randomized distributed computation and can be used for the analysis of nontrivial randomized distributed algorithms. We have identified three forms of compositionality: process compositionality, property compositionality, and feature compositionality. Process and property compositionality are just two new names for forms of compositionality that are widely known in the literature; feature compositionality, although typically used in mathematics, is a form of compositionality that is not usually considered as compositionality. We have shown how feature compositionality plays a crucial role in simplifying the analysis of a randomized system.

An obvious question is whether we have been lucky in our case study and whether the main idea of separating probability from nondeterminism really works. Although we cannot claim that it is always possible to obtain a clean separation between probability and nondeterminism, our experience with the analysis of randomized algorithms [25, 32, 1, 38] gives us a reasonable confidence that a separation can be obtained. The original choice of studying the algorithm of Aspnes and Herlihy [33] was guided mainly by the idea of looking for an algorithm where such separation appeared to be difficult to achieve. Of course, the main problem in the analysis of a system is to understand how to decompose the system, which is still nontrivial.

Another question concerns the generality of the model. In the interaction between two systems we always assume that the probabilistic choices of each component are independent. In other words, it is not possible for a process to condition the probability distribution associated with the transitions of another process (our model is not generative). This restriction limits considerably the field of application of our theory. We understand that such limitations exist, and it would be desirable to overcome them. Unfortunately, we do not know of any way of extending the CSP synchronization style to a model that is not reactive, and on the other hand we find such synchronization mechanism very useful for the analysis of distributed algorithms.

Acknowledgments. I would like to thank the organizers of COMPOS'97 for inviting me to the symposium and for the wonderful exchange environment they have provided.

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. J. Aspnes and M.P. Herlihy. Fast randomized consensus using shared memory. *Journal of Algorithms*, 15(1):441–460, September 1990.
3. J.C.M. Baeten, J.A. Bergstra, and S.A. Smolka. Axiomatizing probabilistic processes: ACP with generative probabilities. In Cleaveland [11], pages 472–485.
4. J.C.M. Baeten and J.W. Klop, editors. *Proceedings of CONCUR 90,* Amsterdam, volume 458 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990.
5. J.C.M. Baeten and W.P Weijland. *Process Algebra*. Cambridge Tracts in Theoretical Computer Science 18, Cambridge University Press, 1990.
6. K.M. Chandi and J. Misra. *Parallel Program Design: A Foundation*. Addison-Wesley, 1988.
7. I. Christoff. Testing equivalences and fully abstract models for probabilistic processes. In Baeten and Klop [4], pages 126–140.
8. I. Christoff. *Testing Equivalences for Probabilistic Processes*. PhD thesis, Department of Computer Science, Uppsala University, 1990.
9. L. Christoff. *Specification and Verification Methods for Probabilistic Processes*. PhD thesis, Department of Computer Science, Uppsala University, 1993.
10. R. Cleaveland, S.A. Smolka, and A. Zwarico. Testing preorders for probabilistic processes (extended abstract). In *Proceedings* 19^{th} *ICALP,* Madrid, volume 623 of *Lecture Notes in Computer Science*, pages 708–719. Springer-Verlag, 1992.
11. W.R. Cleaveland, editor. *Proceedings of CONCUR 92,* Stony Brook, NY, USA, volume 630 of *Lecture Notes in Computer Science*. Springer-Verlag, 1992.
12. S. Dolev, A. Israeli, and S. Moran. Analyzing expected time by scheduler-luck games. *IEEE Transactions on Parallel and Distributed Systems*, 8(4):424–440, April 1997.
13. W. Feller. *An Introduction to Probability Theory and its Applications. Volume 1*. Jokn Wiley & Sons, Inc., 1950.
14. 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.
15. A. Giacalone, C.C Jou, and S.A. Smolka. Algebraic reasoning for probabilistic concurrent systems. In *Proceedings of the Working Conference on Programming Concepts and Methods (IFIP TC2),* Sea of Galilee, Israel, 1990.
16. R.J. van Glabbeek, S.A. Smolka, and B. Steffen. Reactive, generative, and stratified models of probabilistic processes. *Information and Computation*, 121(1):59–80, 1996.

17. H. Hansson. *Time and Probability in Formal Design of Distributed Systems*, volume 1 of *Real-Time Safety Critical Systems*. Elsevier, 1994.
18. C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall International, Englewood Cliffs, 1985.
19. B. Jonsson and J. Parrow, editors. *Proceedings of CONCUR 94,* Uppsala, Sweden, volume 836 of *Lecture Notes in Computer Science.* Springer-Verlag, 1994.
20. R. Keller. Formal verification of parallel programs. *Communications of the ACM*, 7(19):561–572, 1976.
21. E. Kushilevitz and M. Rabin. Randomized mutual exclusion algorithms revisited. In *Proceedings of the* 11^{th} *Annual ACM Symposium on Principles of Distributed Computing,* Quebec, Canada, pages 275–284, 1992.
22. K.G. Larsen and A. Skou. Bisimulation through probabilistic testing. In *Conference Record of the* 16^{th} *ACM Symposium on Principles of Programming Languages,* Austin, Texas, pages 344–352, 1989.
23. K.G. Larsen and A. Skou. Compositional verification of probabilistic processes. In Cleaveland [11], pages 456–471.
24. 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* $8^{\rm th}$ *Annual ACM Symposium on Principles of Programming Languages*, pages 133–138, January 1981.
25. N.A. Lynch, I. Saias, and R. Segala. Proving time bounds for randomized distributed algorithms. In *Proceedings of the* 13^{th} *Annual ACM Symposium on Principles of Distributed Computing,* Los Angeles, CA, pages 314–323, 1994.
26. N.A. Lynch and M.R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proceedings of the* 6^{th} *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.
27. Nancy Lynch and Frits Vaandrager. Forward and backward simulations – Part I: Untimed systems. *Information and Computation*, 121(2):214–233, September 1995.
28. R. Milner. *Communication and Concurrency.* Prentice-Hall International, Englewood Cliffs, 1989.
29. D.M.R. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, 5^{th} *GI Conference*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer-Verlag, 1981.
30. A. Pnueli. On the extremely fair treatment of probabilistic algorithms. In *Proceedings of the* 15^{th} *Annual ACM Symposium on Theory of Computing, Boston, Massachusetts,* May 1983.
31. A. Pnueli and L. Zuck. Verification of multiprocess probabilistic protocols. *Distributed Computing*, 1(1):53–72, 1986.
32. 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.
33. A. Pogosyants, R. Segala, and N. Lynch. Verification of the randomized consensus algorithm of Aspnes and Herlihy: a case study. Technical Memo MIT/LCS/TM-555, MIT Laboratory for Computer Science, 1997.
34. M.O. Rabin. Probabilistic automata. *Information and Control*, 6:230–245, 1963.
35. J.R. Rao. Reasoning about probabilistic algorithms. In *Proceedings of the* 9^{th} *Annual ACM Symposium on Principles of Distributed Computing,* Quebec, Canada, August 1990.
36. I. Saias. Proving probabilistic correctness: the case of Rabin's algorithm for mutual exclusion. In *Proceedings of the* 11^{th} *Annual ACM Symposium on Principles of Dis-*

tributed Computing, Quebec, Canada, August 1992.

37. R. Segala. A compositional trace-based semantics for probabilistic automata. In I. Lee and S.A. Smolka, editors, *Proceedings of CONCUR 95,* Philadelphia, PA, USA, volume 962 of *Lecture Notes in Computer Science*, pages 234–248. Springer-Verlag, 1995.
38. 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.
39. R. Segala and N.A. Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing*, 2(2):250–273, 1995.
40. K. Seidel. Probabilistic communicating processes. Technical Report PRG-102, Ph.D. Thesis, Programming Research Group, Oxford University Computing Laboratory, 1992.
41. K. Seidel, C. Morgan, and A. McIver. An introduction to probabilistic predicate transformers. Technical Report PRG-TR-6-96, Programming Research Group, Oxford University Computing Laboratory, 1996.
42. C. Tofts. A synchronous calculus of relative frequencies. In Baeten and Klop [4].
43. 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.
44. S.H. Wu, S. Smolka, and E.W. Stark. Composition and behaviors of probabilistic I/O automata. In Jonsson and Parrow [19].
45. W. Yi and K.G. Larsen. Testing probabilistic and nondeterministic processes. In *Protocol Specification, Testing and Verification XII*, pages 47–61, 1992.
46. S. Yuen, R. Cleaveland, Z. Dayar, and S. Smolka. Fully abstract characterizations of testing preorders for probabilistic processes. In Jonsson and Parrow [19].
47. L. Zuck. *Past Temporal Logic*. PhD thesis, The Weizman Institute of Science, 1986.