The Essence of Coin Lemmas

Roberto Segala
Dipartimento di Scienze dell'Informazione
Università di Bologna - Italy

Abstract

Coin lemmas are one of the tools for the analysis of randomized distributed algorithms. Their principal role is to reduce the analysis of a randomized system to the analysis of an ordinary nondeterministic system. This paper describes the main ideas behind the formulation and use of coin lemmas and gives examples of coin lemmas of increasing complexity and generality.

1 Introduction

Coin lemmas [8, 10, 11] are one of the existing tools for the analysis of randomized distributed algorithms. An algorithm is modeled as a *probabilistic automaton*, a labeled transition system whose transitions lead to probability distributions over states rather than to single states; a computation of an algorithm is modeled as a *probabilistic execution*, a cycle-free Markov chain obtained by unfolding the transition relation of the algorithm and by choosing a single transition at each point of the unfolding. The operation of choosing one transition at each point of the unfolding is called *resolution of nondeterminism* and is usually seen as an operation under the control of an *adversary*.

Typically a correctness statement of an algorithm asserts that all the *legal* computations of the algorithm satisfy some good property (e.g., a successful action is performed with probability 1 or within expected time t), where by a legal computation we mean a computation that conforms to the environment in which an algorithm runs (e.g., fairness to all the components of a system). The correctness of a randomized algorithm is usually based on the fact that some form of stochastic process takes place in each computation and that some specific results of the stochastic process that is taking place lead to success. Unfortunately, the process of mapping a stochastic process onto the computations of an algorithm has turned out to be very hard and prone to errors [9, 1]. Coin lemmas are one of the tools that allow us to map stochastic processes onto probabilistic executions, possibly without running into traps.

In this paper we discuss the main idea behind coin lemmas and we propose several coin lemmas of increasing complexity and generality. We also compare our coin lemmas with scheduler-luck games [4], another technique proposed independently to solve the same problem. We start with an overview of probabilistic automata.

2 Probabilistic Automata

In this section we describe some of the features of probabilistic automata. The reader interested in more details and in a comparison with other models is referred to [11].

2.1 Probability Spaces

A *probability space* is a triple $(\Omega, \mathcal{F}, P)$ where Ω is a set, $\mathcal{F}$ is a collection of subsets of Ω that is closed under complement and countable union and such that $\Omega \in \mathcal{F}$, and P is a function from $\mathcal{F}$ to $[0,1]$ such that $P[\Omega] = 1$ and 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 $C \subseteq \Omega$, $P[C] = \sum_{x \in C} P[\{x\}]$. Given a set X, denote by $\mathit{Probs}(X)$ the set of discrete probability spaces whose sample space is a subset of X and such that the probability of each element is not 0.

The Dirac distribution over an element x, denoted by $\mathcal{D}(x)$, is the probability space with a unique element x.

Throughout the paper we denote a probability space $(\Omega, \mathcal{F}, P)$ by $\mathcal{P}$. As a notational convention, if $\mathcal{P}$ is decorated with indices and primes, then the same indices and primes carry to its elements. Thus, $\mathcal{P}'_i$ denotes $(\Omega'_i, \mathcal{F}'_i, P'_i)$.

A function $f : \Omega \to \Omega'$ is said to be a *measurable* function from $(\Omega, \mathcal{F})$ to $(\Omega', \mathcal{F}')$ if for each set C of $\mathcal{F}'$ the inverse image of C, denoted by $f^{-1}(C)$, is an element of $\mathcal{F}$. Let P be a probability measure on $(\Omega, \mathcal{F})$, and let P' be defined on $\mathcal{F}'$ as follows: for each element C of $\mathcal{F}'$, $P'(C) = P(f^{-1}(C))$. Then P' is a probability measure on $(\Omega', \mathcal{F}')$. The measure P' is called the measure *induced* by f, and is denoted by $f(P)$. If $\mathcal{P}$ is a discrete probability space and f is a function defined on Ω, then f can be *extended* to $\mathcal{P}$ by defining $f(\mathcal{P})$ to be the discrete probability space $(f(\Omega), 2^{f(\Omega)}, f(P))$.

2.2 Probabilistic Automata

A labeled transition system, also called an *automaton*, is a state machine with labeled transitions. Each transition leaves from a state and leads to the occurrence of a label, also called an *action*, and to a state. A probabilistic automaton is like an ordinary automaton except that each transition leads to an action and to a probability distribution over states.

Resolving the nondeterminism in an automaton leads to a linear chain of states interleaved with actions, called an *execution* or a *computation*; resolving the nondeterminism in a probabilistic automaton leads to a Markov chain structure, since each transition leads probabilistically to more than one state. Such a structure is called a *probabilistic execution*. A probabilistic execution can be visualized as a probabilistic automaton that enables at most one transition from each state (a *fully probabilistic automaton*).

Nondeterminism could be resolved also using randomization: a scheduler for n processes running in parallel could choose the next process to schedule by rolling an n-side dice; similarly, if some actions model the input of an external environment, the environment could provide the input at random or could provide no input with some non-zero probability. Thus, in a probabilistic execution the transition that leaves from a state may lead to a probability distribution over both actions and states and also over deadlock (no input). This new kind of transition is not part of our informal definition of a probabilistic automaton; yet, we find convenient to view a probabilistic execution as a probabilistic automaton.

Thus, the formal definition of a probabilistic automaton allows for a transition to lead to probability distributions over actions and states and over a symbol δ that models deadlock; however, except for the handling of probabilistic executions, we concentrate on *simple probabilistic automata*, which permit only probabilistic choices over states within a transition.

Definition 2.1 A *probabilistic automaton* M consists of four components:

1. a set $states(M)$ of states,
2. a nonempty set $start(M) \subseteq states(M)$ of start states,
3. an action signature $sig(M) = (ext(M), int(M))$ where $ext(M)$ and $int(M)$ are disjoint sets of *external* and *internal* actions, respectively,
4. a transition relation $trans(M) \subseteq states(M) \times Probs((acts(M) \times states(M)) \cup \{\delta\})$, where $acts(M)$ denotes the set $ext(M) \cup int(M)$ of actions, and δ is a special symbol not in $acts(M)$.

A probabilistic automaton M is *simple* if for each transition $(s, \mathcal{P})$ of $trans(M)$ there is an action a such that $\Omega \subseteq \{a\} \times states(M)$. In such a case a transition can be represented alternatively as $(s, a, \mathcal{P}')$, where $\mathcal{P}' \in Probs(states(M))$, and is called a *simple transition*.

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

Observe that an ordinary automaton is a special case of a probabilistic automaton where each transition leads to a Dirac distribution.

2.3 Executions and Probabilistic Executions

We now move to the notion of an execution, which is the result of resolving both the nondeterministic and the probabilistic choices in a probabilistic automaton; it corresponds to the notion of an execution for ordinary automata. We introduce also a notion of an extended execution, which we use later to study the probabilistic behavior of a probabilistic automaton.

Definition 2.2 An *execution* α of a probabilistic automaton M is a (finite or infinite) sequence of alternating states and actions starting with a start state and, if the execution is finite, ending in a state, $\alpha = s_0 a_1 s_1 a_2 s_2 \cdots$, where for each i there exists a probability space $\mathcal{P}$ such that $(s_i, \mathcal{P}) \in trans(M)$ and $(a_{i+1}, s_{i+1}) \in \Omega$. Denote by $fstate(\alpha)$ the first state of α, and, if α is finite, denote by $lstate(\alpha)$ the last state of α. Denote by $exec^*(M)$ and $exec(M)$ the sets of finite and all executions of M, respectively.

An extended execution of M is either an execution of M, or a finite execution of M followed by the symbol δ.

An extended execution α_1 of M is a *prefix* of an extended execution α_2 of M, written $\alpha_1 \leq \alpha_2$, if either $\alpha_1 = \alpha_2$ or α_2 is obtained by extending α_1, i.e., $\alpha_1 = s_0 a_1 s_1 \cdots a_n s_n$ and $\alpha_2 = s_0 a_1 s_1 \cdots a_n s_n a_{n+1} s_{n+1} \cdots$. ■

As we said already, an execution is the result of resolving both the nondeterministic and the probabilistic choices in a probabilistic automaton. The result of the resolution of nondeterministic choices only is a fully probabilistic automaton, called a *probabilistic execution*, which is the entity that replaces the executions of ordinary automata. Informally, since in ordinary automata there is no probability left once the nondeterminism is resolved, the executions and probabilistic executions of an ordinary automaton describe the same objects. Before giving the formal definition of a probabilistic execution, we introduce *combined transitions*, which allow us to express the ability to resolve the nondeterminism using probability. Informally, a combined transition leaving from a state s is obtained by choosing a transition that leaves from s probabilistically, and then behaving according to the transition chosen. Among the choices it is possible not to schedule any transition. This possibility is expressed by the term $(1-\sum_i p_i)$ in the probability of δ in the definition below.

Definition 2.3 Given a probabilistic automaton M, a finite or countable set $\{\mathcal{P}_i\}_i$ of probability distributions of $\mathit{Probs}((\mathit{acts}(M) \times \mathit{states}(M)) \cup \{\delta\})$, and a weight $p_i > 0$ for each i such that $\sum_i p_i \leq 1$, the combination $\sum_i p_i \mathcal{P}_i$ of the distributions $\{\mathcal{P}_i\}_i$ is the probability space $\mathcal{P}$ such that

- $\Omega = \begin{cases} \cup_i \Omega_i & \text{if } \sum_i p_i = 1 \\ \cup_i \Omega_i \cup \{\delta\} & \text{if } \sum_i p_i < 1 \end{cases}$
- $\mathcal{F} = 2^{\Omega}$
- for each $(a, s) \in \Omega$, $P[(a, s)] = \sum_{i|(a,s)\in\Omega_i} p_i P_i[(a, s)]$
- if $\delta \in \Omega$, then $P[\delta] = (1 - \sum_i p_i) + \sum_{i|\delta\in\Omega_i} p_i P_i[\delta]$.

A pair $(s, \mathcal{P})$ is a *combined transition* of M if there exists a finite or countable family of transitions $\{(s, \mathcal{P}_i)\}_i$ and a set of positive weights $\{p_i\}_i$ with $\sum_i p_i \leq 1$, such that $\mathcal{P} = \sum_i p_i \mathcal{P}_i$. Denote $(s, \mathcal{P})$ by $\sum_i p_i(s, \mathcal{P}_i)$. ■

We are now ready to define a probabilistic execution. Informally, a probabilistic execution can be seen as the result of unfolding the transition relation of a probabilistic automaton and then choosing probabilistically a transition from each state.

Definition 2.4 Let α be a finite execution of a probabilistic automaton M. Define a function $\alpha^\frown$ that applied to a pair (a, s) returns $(a, \alpha a s)$, and applied to δ returns δ. Recall from the last paragraph of Section 2.1 that the function $\alpha^\frown$ can be extended to discrete probability spaces.

A *probabilistic execution* of a probabilistic automaton M, is a fully probabilistic automaton, denoted by H, such that

1. $\mathit{states}(H) \subseteq \mathit{exec}^*(M)$. Let q range over states of probabilistic executions.
2. $\mathit{start}(H)$ contains a start state of M.
3. for each transition $tr = (q, \mathcal{P})$ of H there is a combined transition $tr' = (\mathit{lstate}(q), \mathcal{P}')$ of M, called the *corresponding combined transition*, such that $\mathcal{P} = q^\frown \mathcal{P}'$.
4. each state of H is reachable and enables one transition, where a state q of H is reachable if there is an execution of H whose last state is q.

Denote by $\mathit{prexec}(M)$ the set of probabilistic executions of M. Also, for each transition $(q, \mathcal{P})$ of H, denote $\mathcal{P}$ by $\mathcal{P}_q^H$.

There is a strong correspondence between the extended executions of a probabilistic automaton and the extended executions of one of its probabilistic executions. We express this correspondence by means of an operator $\downarrow$ that takes an extended execution α of H and gives back the corresponding extended execution $\alpha\downarrow$ of M, and an operator $\uparrow$ that takes an extended execution α of M and gives back the corresponding extended execution $\alpha\uparrow$ of H if it exists.

2.4 Events

We now define a probability space $(\Omega_H, \mathcal{F}_H, P_H)$ for a probabilistic execution H, so that it is possible to analyze the probabilistic behavior of a probabilistic automaton once the nondeterminism is resolved. The sample space Ω_H is the set of extended executions of M that represent complete extended executions of H, where an extended execution α of H is complete iff it is either infinite or $\alpha = \alpha'\delta$ and $\delta \in \Omega^H_{lstate(\alpha')}$. For each finite extended execution α of M, let C_α, the *cone* with prefix α, be the set $\{\alpha' \in \Omega_H \mid \alpha \leq \alpha'\}$, and let $\mathcal{C}_H$ be the class of cones for H. The probability $\mu_H(C_\alpha)$ of the cone C_α is the product of the probabilities associated with each edge that generates α in H. Formally, if $\alpha = q_0^H a_1 s_1 \cdots s_{n-1} a_n s_n$, then $\mu_H(C_\alpha) \triangleq P^H_{q_0}[(a_1, q_1)] \cdots P^H_{q_{n-1}}[(a_n, q_n)]$, where each q_i is defined to be $q_0^H a_1 s_1 \cdots a_i s_i$, and if $\alpha = q_0^H a_1 s_1 \cdots s_{n-1} a_n s_n \delta$, then $\mu_H(C_\alpha) \triangleq P^H_{q_0}[(a_1, q_1)] \cdots P^H_{q_{n-1}}[(a_n, q_n)] P^H_{q_n}[\delta]$, where each q_i is defined to be $q_0^H a_1 s_1 \cdots a_i s_i$. In [11] it is shown that there is a unique measure $\bar{\mu}_H$ that extends μ_H to the σ-field $\sigma(\mathcal{C}_H)$ generated by $\mathcal{C}_H$, i.e., the smallest σ-field that contains $\mathcal{C}_H$. Then, $\mathcal{F}_H$ is $\sigma(\mathcal{C}_H)$ and P_H is $\bar{\mu}_H$. With this definition it is possible to show that any union of cones is measurable.

3 The Idea Behind Coin Lemmas

We illustrate coin lemmas through some examples of problems and algorithms.

Example 3.1 (Dining Philosophers) There are n philosophers sitting at a round table with a bowl of spaghetti in the center. Each philosopher has a fork on his left and another fork on his right. The left fork is shared with the left neighbor and the right fork is shared with the right neighbor. Sometimes a philosopher decides to eat and does it by picking up his two forks, one at a time. No philosopher can eat without picking up his two forks first. The problem is to ensure that if some philosopher wants to eat, then eventually some philosopher will eat.

It is known from [7] that there is no symmetric solution to the dining philosopher's problem, i.e., there is no solution where all philosophers follow the same algorithm. In [7] Lehmann and Rabin propose the following randomized algorithm: a philosopher who wants to eat flips a coin to choose which fork to pick up first, waits for the chosen fork to be free and picks it up, and finally tries to pick up the other fork. If the other fork is free, then the philosopher eats; otherwise the philosopher puts down the first fork and starts again from the beginning.

The main idea behind the algorithm of Lehmann-Rabin is that whenever two adjacent philosophers flip coins and none of them chooses the shared fork, then one of the two philosophers will find the second fork free. ■

Example 3.2 (Consensus in Exponential Time) There are n processes that propose a value in the set $\{0,1\}$. The values proposed by the processes may be different, but at the end all the processes must agree on the same value, chosen among the values that were proposed. Communication is asynchronous and processes may crash by stopping. This problem is unsolvable [6] since roughly it is not possible to distinguish between a crashed process and a slow process.

A randomized algorithm that solves the consensus problem works as follows [3]. The algorithm is structured in rounds. At each round the processes interact and try to produce a consistent value to agree on. If the processes cannot agree, then they flip a coin (a different coin for each process) to choose the value to propose in the next round. With probability $1/2^n$ all processes choose the same value, and then agreement is possible. ■

Example 3.3 (Consensus in Polynomial Time) The algorithm described in Example 3.2 works in expected exponential time since there is an exponentially low probability that randomization leads to agreement. In [2] a different way to flip coins is proposed so that the probability to reach agreement is constant. The processes that need to flip a coin flip local coins to increment or decrement a shared counter. When the value of the counter goes beyond some fixed barriers (with values $\pm K$), then the processes return a value. The value of the shared counter evolves like a process known as *random walk* [5], and it is possible to use random walk theory to show that there is a constant probability that all processes observe values beyond the same barrier, i.e., all processes return the same value. ■

Example 3.4 (Choosing a Leader) A randomized algorithm to choose a leader among n processes can be structured in rounds as follows. At each round there are some processes, determined by race conditions, that participate in a game. In the game each process flips coins until a head comes out and returns the number of coins that were flipped, i.e., number i is chosen with probability $1/2^i$. The winners are the processes that draw the highest number. If there is a unique winner, then the winner is the leader.

It is known that if k processes participate in the game, where k is any fixed number, then there is a constant probability that there is a unique winner. The constant is independent of k. Thus, at each round there is constant probability to elect a leader, and a leader is elected within an expected constant number of rounds. ■

In each of the examples above the correctness of the related algorithm is based on the intuition that at some point the algorithm behaves like a specific stochastic process and that some of the results of the stochastic process lead to success. In Example 3.1 the process consists of flipping two coins, each one giving a specified value; in Example 3.2 the process consists of flipping n coins, all giving the same value; in Example 3.3 the process consists of computing a random walk where a specific barrier is reached before another specific barrier; in Example 3.4 the process consists of drawing k numbers where there is a unique maximum.

To be sure that our arguments are correct, we need to verify that it is possible to identify the chosen stochastic processes in each legal computation of the algorithm and that in each computation the chosen events guarantee the success of the algorithm. Furthermore, we need to verify that the probabilities of the events of our interest are preserved in the mapping.

Although these operations may appear to be simple, Example 3.4 provides us with several problems. If we consider a computation of the algorithm and we focus on one round, we observe that some processes, say k, participate in the game. So, we map the game with k participants onto the computation. However, a computation is a Markov chain, and it may not be the case that there are k participants in each branch of the chain. What are we supposed to do with the branches with less than k participants? This detail was overlooked in the original analysis of the protocol of Example 3.4, and perhaps we have been good enough to induce the reader into the same kind of mistake in our informal description of the analysis. Indeed, the algorithm of Example 3.4 does not work. We said that the players of the game are determined by race condition, but we have not given any constraint on how race conditions determine the players. Thus, an adversary could start with two players and add players until either there are two winners or there are no more players available. In this case the probability of a unique winner is negligible.

Similar problems, although not so catastrophic, occur in the other three examples. What happens if no two adjacent processes flip a coin in the randomized dining philosophers algorithm? What happens if not all processes flip a coin in the consensus algorithm of Example 3.2? What happens if none of the barriers is reached in the random walk of Example 3.3 due to the fact that all processes stop flipping coins? In the first case no two neighbor philosophers want to eat, and thus each philosopher desiring to eat finds both forks free and succeeds; in the second case not all processes disagree and it is sufficient that the other processes choose consistent values; in the third case all the processes involved in the coin flipping process crash. The difference between these last three examples and Example 3.4 is that in the last three examples all the situations where some coins are not flipped are successful. More precisely, all the situations where it is possible to choose values for the unflipped coins so that the stochastic process we have in mind is successful are also successful for the algorithm.

The purpose of coin lemmas is to ensure that we do not miss any of the dangerous cases where some coins may not be flipped. Given a stochastic process, a coin lemma provides us with two objects.

1. A rule to choose an event in each probabilistic execution of a probabilistic automaton;
2. A lower bound on the probability of the events identified by the rule.

The rule is mechanical. Given a probabilistic execution there is a well specified way to identify the places where the stochastic process is taking place and to identify the related results. If some experiments of the stochastic process cannot be identified (e.g., one of the coins is not flipped), then the rule takes the point that the related experiment is successful. That is, given an execution of a probabilistic execution, the execution is in the event identified by the rule if it is possible to choose results for the experiments that did not take place so that the results of the experiments that did take place together with the chosen results are one of the successful results of the stochastic process under consideration. With such a rule the corresponding lower bound on the probability of the identified events is the probability of the event under consideration in the starting stochastic process.

The error with Example 3.4 is then identified immediately. All the branches with less than k participants can be completed to games with a unique winner, and thus are part of the event identified by the rule. However, there may be several branches with less than k participants with more than one winner.

In the next two sections we present several specific coin lemmas to illustrate better the main idea behind a coin lemma.

4 Some Simple Coin Lemmas

We present some simple coin lemmas of increasing complexity where the rule to identify an event is based on actions. We prove only some significative lemmas.

The first event that we consider is a simple binary experiment associated with the first occurrence of an action a. The set of successful states is identified by a set U. If p is a lower bound on the probability of reaching a state from U in the transitions labeled by a, then p is also a lower bound on the probability of the event identified by the rule. Observe that all the executions where a does not occur are considered as successful.

Lemma 4.1 *Let M be a simple probabilistic automaton, and let (a, U) be a pair consisting of an action of M and a set of states of M. Let p be a real number between* 0 *and* 1 *such that for each transition $(s, a, \mathcal{P})$, $P[U] \geq p$.*

For each probabilistic execution H of M let $FIRST(a, U)(H)$ be the set of executions α of Ω_H such that either a does not occur in α, or a occurs in α and the state reached after the first occurrence of a is a state of U.

Then, for each probabilistic execution fragment H of M, $P_H[FIRST(a, U)(H)] \geq p$.

Proof: For convenience denote $FIRST(a, U)(H)$ by E, and for each state q of H, denote by $\Omega(q, \overline{U})$ the set $\{(a, q') \in \Omega_q^H \mid lstate(q') \notin U\}$. Let Θ be the set of states q of H such that action a does not occur in q, and $P_q^H[a] > 0$. Then,

$$P_H[\overline{E}] = \sum_{q \in \Theta} \sum_{(a,q') \in \Omega(q,\overline{U})} P_H[C_q] P_q^H[(a, q')]. \tag{1}$$

By expressing $P_q^H[(a, q')]$ as a conditional probability and rearranging the expression, we obtain

$$P_H[\overline{E}] = \sum_{q \in \Theta} P_H[C_q] P_q^H[a] \left(\sum_{(a,q') \in \Omega(q,\overline{U})} P_q^H[(a, q')|a] \right). \tag{2}$$

From the definition of a probabilistic execution fragment and the definition of $\Omega(q, \overline{U})$, for each element q of Θ there is a combined transition $tr = \sum_i p_i tr_i$ of M such that $tr_q^H = q \frown tr$ and

$$\sum_{(a,q') \in \Omega(q,\overline{U})} P_q^H[(a, q')|a] = P_{tr}[\overline{U}|a] = \frac{P_{tr}[\overline{U} \cap a]}{P_{tr}[a]} = \frac{\sum_i p_i P_{tr_i}[\overline{U} \cap a]}{\sum_i p_i P_{tr_i}[a]}. \tag{3}$$

By multiplying and dividing each i^{th} summand of the enumerator by $P_{tr_i}[a]$, using the hypothesis of the lemma, i.e., for each i $P_{tr_i}[\overline{U} \cap a] \leq (1-p)$, and simplifying algebraically, from (3) we obtain

$$\sum_{(a,q') \in \Omega(q,\overline{U})} P_q^H[(a, q')|a] \leq (1 - p). \tag{4}$$

By using (4) in (2) we obtain

$$P_H[\overline{E}] \leq (1-p)\left(\sum_{q\in\Theta} P_H[C_q]P_q^H[a]\right). \tag{5}$$

Furthermore, the subexpression $\sum_{q\in\Theta} P_H[C_q]P_q^H[a]$ is the probability that a occurs in H, which is at most 1. Thus,

$$P_H[\overline{E}] \leq (1-p). \tag{6}$$

This completes the proof. ∎

The events of the form $FIRST(a, U)$ can be generalized to account for the first action that occurs among several possible actions. Let M be a probabilistic automaton, and let $(a_1, U_1), \ldots, (a_n, U_n)$ be pairs consisting of an action of M and a set of states of M such that the actions a_i are all distinct. Then, for each probabilistic execution H of M, define $FIRST((a_1, U_1), \ldots, (a_n, U_n))(H)$ to be the set of executions α of Ω_H such that either none of the a_i's occurs in α, or some of the a_i's occur in α, and if a_i is the first of those actions that occurs, then the state reached after the first occurrence of a_i is a state of U_i.

It is simple to check that $FIRST((a_1, U_1), \ldots, (a_n, U_n))(H)$ is an event since the complement of $FIRST((a_1, U_1), \ldots, (a_n, U_n))(H)$ can be expressed as a union of cones.

Lemma 4.1 extends to this case.

Lemma 4.2 *Let M be a simple probabilistic automaton, and let $(a_1, U_1), \ldots, (a_n, U_n)$ be pairs consisting of an action of M and a set of states of M such that the actions a_i are all distinct. Let $\{p_i\}_{i=1,\ldots,n}$ be a collection of real numbers between 0 and 1 such that for each i, $1 \leq i \leq n$, and each transition $(s, a_i, \mathcal{P})$ of M, $P[U] \geq p_i$. Then, for each probabilistic execution fragment H of M, $P_H[FIRST((a_1, U_1), \ldots, (a_n, U_n))(H)] \geq min(p_1, \ldots, p_n)$.*

Proof: Let V be $\{a_1, \ldots, a_n\}$, and let p be the minimum of $\{p_1, \ldots, p_n\}$. For convenience, denote $FIRST((a_1, U_1), \ldots, (a_n, U_n))(H)$ by E, and for each state q of H, denote by $\Omega(q, \overline{E})$ the set $\cup_{i\in\{1,\ldots,n\}}\{(a_i, q') \in \Omega_q^H \mid lstate(q') \notin U_i\}$. Then, for each transition $(q, \mathcal{P}_q^H)$ of H such that $P_q^H[V] > 0$,

$$P_q^H[\Omega(q, \overline{E})|V] \leq (1-p). \tag{7}$$

To prove (7), let, for each $i = 1, \ldots, n$, $\Omega(q, a_i, \overline{U}_i)$ denote the set $\{(a_i, q') \in \Omega_q^H \mid lstate(q') \notin U_i\}$. Then,

$$P_q^H[\Omega(q, \overline{E})|V] = \sum_{i\in\{1,\ldots,n\}} P_q^H[\Omega(q, a_i, \overline{U}_i)|V]. \tag{8}$$

By using conditional probabilities, Equation (8) can be rewritten into

$$P_q^H[\Omega(q, \overline{E})|V] = \sum_{i\in\{1,\ldots,n\}} P_q^H[a_i|V]P_q^H[\Omega(q, a_i, \overline{U}_i)|a_i]. \tag{9}$$

Following the same argument as in the proof of Lemma 4.1, for each i, $P_q^H[\Omega(q, a_i, \overline{U}_i)|a_i] \leq (1-p)$; moreover, $\sum_i P_q^H[a_i|V] = 1$. Thus, (7) follows directly.

The rest of the proof follows te lines of the proof of Lemma 4.1. Let Θ be the set of states q of H such that no action of V occurs in q, and $P_q^H[V] > 0$. Then,

$$P_H[\overline{E}] = \sum_{q \in \Theta} \sum_{(a,q') \in \Omega(q,\overline{E})} P_H[C_q] P_q^H[(a,q')]. \tag{10}$$

By expressing $P_q^H[(a,q')]$ as a conditional probability and rearranging the expression, we obtain

$$P_H[\overline{E}] = \sum_{q \in \Theta} P_H[C_q] P_q^H[V] \left(\sum_{(a,q') \in \Omega(q,\overline{E})} P_q^H[(a,q')|V] \right). \tag{11}$$

The subexpression $\sum_{(a,q') \in \Omega(q,\overline{E})} P_q^H[(a,q')|V]$ is $P_q^H[\Omega(q,\overline{E})|V]$, which is less than or equal to $(1-p)$ from (7). Thus,

$$P_H[\overline{E}] \leq (1-p) \left(\sum_{q \in \Theta} P_H[C_q] P_q^H[V] \right). \tag{12}$$

Furthermore, the subexpression $\sum_{q \in \Theta} P_H[C_q] P_q^H[V]$ is the probability that an action from V occurs in H, which is at most 1. Thus,

$$P_H[\overline{E}] \leq (1-p). \tag{13}$$

This completes the proof. ■

In the definition of *FIRST* we have considered the first action among a given set that occurs in a probabilistic execution fragment H. However, the results for *FIRST* are valid also if we consider the i^{th} occurrence of an action instead of the first occurrence. This observation suggests a new more general collection of events.

Let M be a probabilistic automaton, and let $(a_1, U_1), \ldots, (a_n, U_n)$ be pairs consisting of an action of M and a set of states of M such that the actions a_i are all distinct. Then, for each probabilistic execution H of M, define $OCC(i, (a_1, U_1), \ldots, (a_n, U_n))(H)$ to be the set of executions α of Ω_H such that either there are less than i occurrences of actions from $\{a_1, \ldots, a_n\}$ in α, or there are at least i occurrences of actions from $\{a_1, \ldots, a_n\}$, and, if a_j is the action that occurs as the i^{th} one, then the state reached after its occurrence is a state of U_i.

Since in the proof of Lemma 4.2 we never use the fact that it is the first occurrence of an action that is considered, Lemma 4.2 carries over to the i^{th} occurrence trivially.

Lemma 4.3 *Let M be a simple probabilistic automaton, and let $(a_1, U_1), \ldots, (a_n, U_n)$ be pairs consisting of an action of M and a set of states of M such that the actions a_i are all distinct. Let $\{p_j\}_{j=1,\ldots,n}$ be a collection of real numbers between 0 and 1 such that for each $j \in \{1, \ldots, n\}$ and each transition $(s, a_j, \mathcal{P})$ of M, $P[U_j] \geq p_j$. Then, for each probabilistic execution fragment H of M, $P_H[OCC(i, (a_1, U_1), \ldots, (a_n, U_n))(H)] \geq min(p_1, \ldots, p_n)$.* ■

It is also possible to consider the conjunction of several properties as well. In order to simplify the notation, we consider only event schemas of the kind $OCC(i, (a, U))$ since, as we have seen in the proof of Lemma 4.2, the case with multiple actions can be reduced to the case with a single action. The next lemma states that the lower bound on the probability of the conjunction of several event of the form $OCC(i,(a,U))$ is the product of the lower bounds of all the $OCC(i,(a,U))$ events. In other words, an adversary can introduce dependencies by increasing the probability of the conjunction of events, but it can never decrease the probability below the value that we would get by considering all the events to be independent.

Lemma 4.4 *Let M be a simple probabilistic automaton, and let $(k_1,a_1,U_1),\ldots,(k_n,a_n,U_n)$ be a collection of triplets consisting of a natural number, an action of M and a set of states of M, such that the pairs (k_i,a_i) are all distinct. Let $\{p_j\}_{j=1,\ldots,n}$ be a collection of real numbers between 0 and 1 such that for each $j \in \{1,\ldots,n\}$ and each transition $(s,a_j,\mathcal{P})$ of M, $P[U_j] \geq p_j$. Then, for each probabilistic execution fragment H of M, $P_H[OCC(k_1,(a_1,U_1))(H) \cap \cdots \cap OCC(k_n,(a_n,U_n))(H)] \geq p_1 \cdots p_n$.* ∎

If we consider the process of flipping n coins, then the coin lemmas seen so far can be used to study the probability that all the coins yield head. However, we may be interested in the probability that at least half of the coins yield head, or in the probability that exactly 5 coins yield head. Suppose now that want to express the process of rolling a dice. Again, the coin lemmas seen so far are not adequate since they can deal only with binary outcomes: we can observe only whether a specific set U is reached or not. How can we express the event that for each number i between 1 and 6 there is at least one dice that rolls i?

We now describe a coin lemma that can deal with the scenarios outlined above. Let M be a probabilistic automaton, and let $\mathcal{S}$ be a set of n tuples $\{x_1,\ldots,x_n\}$, where for each i, $1 \leq i \leq n$, x_i is a tuple $(a_i,U_{i,1},\ldots,U_{i,k})$ consisting of an action of M and k pairwise disjoint sets of states of M. Let the actions a_i be all distinct. Let E be a set of tuples $((1,j_1),\ldots,(n,j_n))$ where for each i, $1 \leq i \leq n$, the value of j_i is between 1 and k. For each extended execution α of M and each i, $1 \leq i \leq n$, let

$$U_i(\alpha) = \begin{cases} \{(i,1),\ldots,(i,k)\} & \text{if } a_i \text{ does not occur} \\ \{(i,j)\} & \text{if } a_i \text{ occurs and its first occurrence leads to } U_{i,j} \\ \emptyset & \text{otherwise.} \end{cases}$$

For each probabilistic execution H of M, define $GFIRST(\mathcal{S},E)(H)$ to be the set of extended executions α of Ω_H such that $E \cap (U_1(\alpha) \times \cdots \times U_k(\alpha)) \neq \emptyset$.

We illustrate the definition above by encoding the dice rolling example. In each tuple $(a_i,U_{i,1},\ldots,U_{i,k})$ a_i identifies the action of rolling the i^{th} dice, $k=6$, and for each j, $U_{i,j}$ is the set of states where the i^{th} dice rolls j. The set E identifies the set of outcomes that are considered to be good. In the case of the dices E is the set of tuples $((1,j_1),\ldots,(n,j_n))$ where for each number l between 1 and 6 there is at least one i such that $j_i = l$. The function $U_i(\alpha)$ checks whether the i^{th} dice is rolled and identifies the outcome. If the dice is not rolled, then, we allow any outcome as a possible one; if the dice is rolled and hits $U_{i,j}$, then the outcome is (i,j); if the the dice is rolled and the outcome is not in any one of the sets $U_{i,j}$'s, then there is no outcome (this case does not arise in our example). Then, an extended execution α of Ω_H is in the event $GFIRST(\mathcal{S},E)(H)$ if at least one of the outcomes associated with α is an element of E, i.e., if by choosing the outcome of the dices that are not rolled in α all the six numbers appear as the outcome of some dice.

Let p be the probability that by rolling n dices all the six numbers appear as the outcome of some dice. Then, the lemma below states that $P_H[GFIRST(\mathcal{S},E)(H)] \geq p$ for each H.

Proposition 4.5 *Let M be a simple probabilistic automaton. Let $\mathcal{S}$ be a set of n tuples $\{x_1, \ldots, x_n\}$ where for each i, $1 \leq i \leq n$, x_i is a tuple $(a_i, U_{i,1}, \ldots, U_{i,k})$ consisting of an action of M and k pairwise disjoint sets of states of M. Let the actions a_i be all distinct. Let E be a set of tuples $((1, j_1), \ldots, (n, j_n))$ where for each i, $1 \leq i \leq n$, the value of j_i is between 1 and k. For each i, j, $1 \leq i \leq n$, $1 \leq j \leq k$, let $p_{i,j}$ be a real number between 0 and 1 such that for each transition $(s, a_i, \mathcal{P})$ of M, $P[U_{i,j}] \geq p_{i,j}$, and let $\mathcal{C}$ be the collection of the $p_{i,j}$s. Let $P_{\mathcal{C}}[E]$ be the probability of the event E assuming that each experiment i is run independently, and that for each i a pair (i, j) is chosen with probability $p_{i,j}$. Then, for each probabilistic execution fragment H of M, $P_H[GFIRST(\mathcal{S}, E)(H)] \geq P_{\mathcal{C}}[E]$.* ■

5 A General Coin Lemma

In Section 4 we have presented several specific coin lemmas. In this section we present another more general coin lemma, still based on actions, that captures the idea behind the coin lemmas of Section 4. We consider the experiment of drawing a finite sequence of natural numbers.

Let $\Omega = N^m$, where m is a fixed natural number, and let $\mathcal{F} = 2^{\Omega}$. For each finite sequence x of natural numbers with length less than m, let $\mathcal{P}_x$ be the distribution of the element that follows x.

Let M be a simple probabilistic automaton. For each k, $0 \leq k < m$, and each $x \in N^k$, let A_x be a set of tuples $(\alpha, a, U_0, U_1, \ldots)$ where α is a finite execution fragment of M, a is an action of M, and $U_0, U_1, \ldots$ is a partition of the states of M. The occurrence of a after α is the signal that the experiment for the element following x is taking place, and the sets of states $U_1, U_2, \ldots$ represent the results of the experiment.

Suppose that for each x there are no two tuples $(\alpha, a, \ldots)$ and $(\alpha', a', \ldots)$ in A_x such that $\alpha a \leq \alpha'$ and that there are no two x, y with $x < y$ and no pair α, a such that tuples of the form $(\alpha, a, \ldots)$ are both in A_x and A_y. In other words, no experiment occurs twice in an execution and no transition is used for two experiments in the same sequence (all experiments are independent).

Finally, suppose that for each x, each tuple $(\alpha, a, U_1, U_2, \ldots)$ in A_x, each transition $(\mathit{lstate}(\alpha), a, \mathcal{P}_a)$ of M, and each $i \geq 0$, $P_a[U_i] = P_x[i]$.

Let E be an event in $\mathcal{P}$. For each execution fragment α of M, let $x(\alpha)$ be a sequence of length m of elements from $N \cup \{*\}$ such that, for each i, if there is a prefix $\alpha' as$ of α, a tuple $(\alpha', a, U_1, U_2, \ldots)$ of $A_{x_{<i}}$, and a number j such that $s \in U_j$, then $x_i = j$, otherwise $x_i = *$. For a probabilistic execution fragment H of M, let $E(H)$ be the set of elements α of $\mathcal{F}_H$ such that there is an element of E that coincides with $x(\alpha)$ in all the non-$*$ places. Then, $P_H[E(H)] \geq P[E]$.

6 Scheduler-Luck Games

In [4] a technique called scheduler-luck games is presented for the analysis of randomized distributed algorithms. It is assumed that all random draws are uniform and binary; the technique can be extended easily, though. An algorithm is studied as a game between a player *scheduler*, that schedules new transitions, and a player *luck*, that fixes the value of some coins. We say that luck has a k-winning strategy if luck can guarantee the success of the algorithm against any scheduler by fixing the value of at most k coins. In such case the algorithm works correctly with probability at least $1/2^k$.

Scheduler-luck games can be seen as an instance of coin lemmas, where all random draws are binary and the event to consider is all heads. The tuples that are used to identify the strings $x(\alpha)$ are defined so that $x(\alpha)$ is all heads only when the random draws give the results selected by luck. Of course there are cases where scheduler-luck games give a more natural way of analyzing an algorithm, and it is up to the verifier to choose the technique that fits best.

7 Concluding Remarks

We have described a technique for the analysis of randomized distributed algorithms that takes care of the most subtle problem with randomization: making sure that the probabilistic behavior of a randomized algorithm coincides with the expected probabilistic behavior. The technique, named coin lemmas, consists of a mechanical rule to associate events with the computations of an algorithm and a lower bound on the probability of the chosen events. The main advantage of coin lemmas is that the analysis of a randomized distributed algorithm is reduced to the analysis of a system that does not involve probability at all, which can be done using existing techniques.

Although we have given a general coin lemma in this paper, there are many other more general coin lemmas that can be proposed. Unfortunately, the more general a coin lemmas is, the more complex is its statement. Sometimes, as is the case for scheduler-luck games, a coin lemma can be formulated in a natural way using concepts like game theory. It is important to isolate other possible formulations for coin lemmas.

In this paper we have tried to identify what we call the essence of coin lemmas, that is the idea that we need to be explicit on the kind of stochastic process that is taking place and that it is important to prove that all the experiments that may not occur within a computation are favorable to the success of an algorithm. In other words, an adversary should never be advantaged by not flipping coins.

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

[4] 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.

[5] W. Feller. *An Introduction to Probability Theory and its Applications. Volume 1.* Jokn Wiley & Sons, Inc., 1950.

[6] 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.

[7] D. Lehmann and M. Rabin. On the advantage of free choice: a symmetric and fully distributed solution to the dining philosophers problem. In *Proceedings of the 8^{th} Annual ACM Symposium on Principles of Programming Languages*, pages 133–138, January 1981.

[8] 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.

[9] N.A. Lynch and R. Segala. A comparison of simulation techniques and algebraic techniques for verifying concurrent systems. In *Proceedings of the Second North American Process Algebra Workshop, Cornell University, NY*, 1993. An extended version appears as technical memo MIT/LCS/TM-499.

[10] A. Pogosyants and R. Segala. Automatic verification of time properties of randomized distributed algorithms. Manuscript in progress, 1995.

[11] R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems.* PhD thesis, MIT, Dept. of Electrical Engineering and Computer Science, 1995. Also appears as technical report MIT/LCS/TR-676.