URL: http://www.elsevier.nl/locate/entcs/volume22.html 20 pages

The Essence of Coin Lemmas

Roberto Segala

Dipartimento di Scienze dell'Informazione,
Università di Bologna - Italy

Abstract

Coin lemmas are a tool 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

The analysis of randomized distributed algorithms is known to be a hard task [27,32] due to the interactions between probability and nondeterminism. An evidence of this fact is also the existence of several incorrect algorithms in the literature, some of which have been corrected [23,1].

One way of analyzing an algorithm is to let an *adversary* resolve the nondeterminism, identify the worst possible adversary, and show that the algorithm works properly in the presence of the worst adversary. Unfortunately, the identification of the worst adversary is not an easy task and is usually driven by intuition rather than by rigorous analysis.

An alternative approach consists of showing that under any adversary the algorithm works properly. That is, given an arbitrary adversary, show that with a sufficiently high probability the algorithm completes its task successfully (we say that the algorithm is successful with a sufficiently high probability). In this case the problem of identifying the worst adversary disappears; however, it is not clear how to prove that the algorithm is successful with a sufficiently high probability.

Fortunately, the designer of an algorithm knows the reasons for which the algorithm is supposed to work: usually there is some stochastic process taking place during the evolution of the algorithm, and some specific results of the underlying stochastic process guarantee success. However, one main question is left open: how do we make sure that the anticipated stochastic process is really taking place under any adversary? Indeed, nondeterminism creates a lot of interference, and the probability distributions of the underlying

stochastic process could be highly affected. The distortions introduced by nondeterminism are one of the main pitfalls behind incorrect algorithms.

In this context coin lemmas [28,33,37] were introduced. A coin lemma is a rule to map a stochastic process onto events in the computations of a randomized algorithm so that some minimum probability requirements are guaranteed. The advantages of coin lemmas are twofold: on one side they allow us to reduce the problem of analyzing a randomized distributed algorithm to the problem of analyzing an ordinary distributed algorithm, thus removing probability; on the other side coin lemmas force the designer to view an algorithm in such a way that the subtleties due to nondeterminism can be identified easily. It is important to note that there is no formal statement of what is a coin lemma. Rather, the term "coin lemma" is a generic name for the task of mapping stochastic processes onto events in computations while guaranteeing some minimum probability requirement. The reason for the word "coin" is that within randomized distributed algorithms the probabilistic arguments are usually based on coin flipping; the reason for the word "lemma" is that a coin lemma is an auxiliary statement in the context of the analysis of an algorithm. The aim of this paper is to give an idea of how coin lemmas are used and what are the key ideas for the formulation of a new coin lemma. The reader interested in complete applications of coin lemmas is referred to [37,28,33,34].

Coin lemmas were first proposed to be used in conjunction with the *simulation method* [29,30] on Labeled Transition Systems (LTSs) [22], which we also refer to as *automata*. For this reason, coin lemmas are formulated within the framework of Probabilistic Automata (PA) [37], a model with a structure similar to Labeled Concurrent Markov Chains (LCMCs) [39,16] and Markov Decision Processes (MDPs) [10] that at the same time can be seen as a probabilistic extension of LTSs. The advantages of such a formulation are the possibility of defining a compositional theory of probabilistic systems, and the availability of a framework where the reduction from probability to nondeterminism carried out by a coin lemma can be stated easily.

A *probabilistic automaton* is like an LTS with the difference that a transition leads to a discrete probability distribution over states rather than to single states. Thus, an ordinary LTS is a probabilistic automaton where all distributions are Dirac distributions. Probabilistic automata are very similar to MDPs, with the main technical difference that within MDPs it is not possible to choose nondeterministically between two transitions with the same label. Resolving the nondeterminism within a PA amounts to choosing a transition from each state. Within distributed algorithms nondeterminism is resolved by an entity called an *adversary*; within MDPs nondeterminism is resolved through *policies*. In all cases the result of the resolution of the nondeterminism is a cycle free Markov chain whose states record the past history. These objects are called *probabilistic executions* within PAs, and, in accordance with the style of analysis typical for LTSs, are the main objects of our analysis.

We have now enough information to describe better the role of a coin lemma. Given a randomized algorithm, we would like to say that for each probabilistic execution of the algorithm, possibly satisfying some fairness requirements, the probability that the algorithm is successful is at least some value p. Of course we assume that an algorithm is represented as a PA, which we refer to by $\mathcal{A}$. We know, or rather we have a strong intuition, that some stochastic process $\mathcal{S}$ is taking place within each probabilistic execution of the algorithm, and we suspect that whenever the stochastic process $\mathcal{S}$ gives some specific results (successful results), which have probability p, the algorithm is successful. A coin lemma for $\mathcal{S}$ provides us with a rule to map each probabilistic execution of $\mathcal{A}$ onto an event (a set of paths in the probabilistic execution) that has probability at least p. A path in a probabilistic execution coincides with the notion of execution of LTSs. Thus, the problem is reduced to verifying that all the executions that belong to some event obtained from the coin lemma guarantee the success of the algorithm. Our main problem is to see how to formulate a coin lemma; once a coin lemma is given, the verification task is simple.

The first step for the formulation of a coin lemma for a stochastic process $\mathcal{S}$ is to identify the points of an execution in which the elementary experiments of $\mathcal{S}$ take place. In our treatment we use the labels associated with transitions to identify the elementary experiments, although this is not the only possibility. Then, one possible rule is to consider all those paths of a probabilistic execution where the result of $\mathcal{S}$ is one of the successful results. Unfortunately, this rule does not work since sometimes nondeterminism can be resolved in such a way that some of the elementary experiments of $\mathcal{S}$ do not take place. We call a path (an execution) *incomplete* if some elementary experiments of $\mathcal{S}$ do not take place. If we do not include the incomplete paths in our events, then the lower bound p does not hold, since in the worst case it could be possible to resolve the nondeterminism so that some elementary experiment is not scheduled with an arbitrarily high probability; if we include all the incomplete paths, then the lower bound p is satisfied, but the events obtained from the rule could be unnecessarily too large. Our main observation is that it is sufficient to include those incomplete paths for which it is possible to fix the results of the experiments that do not take place and obtain one of the successful results of $\mathcal{S}$. From the point of view of the analysis of randomized distributed algorithms the style of rules that we have just outlined forces the user either to verify that all the elementary experiments take place, or to consider explicitly the cases where some experiments do not take place. Overlooking this last point is the main cause of errors that we have noted.

For the purpose of this paper we work with a simplified probabilistic automaton model where nondeterminism is resolved without using randomization. The drawback of our choice is that in the simplified model it is not possible to reason compositionally. However, compositional reasoning is not relevant for the study of coin lemmas. The reader interested in the full PA

model, where both the results of this paper and a compositional theory are available, is referred to [37].

The rest of the paper is organized as follows. Section 2 gives an overview of related work; Section 3 introduces probabilistic automata and the related concepts that are relevant for this paper; Section 4 illustrates the use and formulation of coin lemmas through several examples; Section 5 gives examples of coin lemmas of increasing complexity and generality; Section 6 gives some concluding remarks.

2 Related Work

There is a lot of work on modeling randomized concurrent systems with process algebras, part of it dealing with nondeterminism [15,24–26,14,21,38,3,40,7,6,8], and part of it dealing with some form of nondeterminism [41,19,17,18]. The work in [17,18] is based on LCMCs. The main difference between LCMCs and PAs is that in LCMCs there is a strict alternation between states that enable a single probabilistic transition (probabilistic states) and states that enable several nondeterministic transitions (nondeterministic states). In our opinion MDPs [10] provide the right foundation for modeling concurrent probabilistic systems. The main difference between MDPs and PAs is that PAs are formulated as a conservative extension of LTSs and enable modular reasoning together with the verification techniques associated with LTSs (e.g., the simulation method). A model with the same structure as PAs was introduced in [35], although with the sole purpose of studying languages. Other models similar to PAs can be found in [20,32].

A verification tool similar to our coin lemmas are the scheduler-luck games of [11]. The rule to identify an event is presented as a game between two players: *scheduler*, who decides what transitions to schedule next, and *luck*, who decides the outcome of the coin flips. We are guaranteed that the resulting events have probability at least $1/2^k$ if luck moves at most k times during the game. Scheduler-luck games can be seen as an instance of a coin lemma. In the cases where they are applicable they provide a nice way of analyzing an algorithm.

This paper is concerned with the analysis of algorithms by hand. Another area of research is concerned with automatic verification, mainly based on probabilistic model checking, first introduced in [39]. Within model checking coin lemmas are not relevant since the proofs are completely automatic. The drawback of probabilistic model checking is that it is applicable only to finite state systems and that its complexity is very high, at the point of not being applicable to large algorithms. Coin lemmas could become relevant if we study hybrid proof techniques where only some parts of a proof are carried out automatically.

Other work si concerned on extending temporal logic based deductive tools to the analysis of randomized concurrent systems. Relevant work includes

[32,42,36,31]. There has been no investigation yet on how coin lemmas could be relevant and/or integrated with such verification techniques.

3 Probabilistic Automata

3.1 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 $Probs(X)$ 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 $f : \Omega \to \Omega'$ can be lifted to discrete probability spaces as follows: $f(\mathcal{P}) = (f(\Omega), 2^{f(\Omega)}, P')$, where, for each $E \subseteq f(\Omega)$, $P'[E] = P[f^{-1}(E)]$.

3.2 Probabilistic Automata

A labeled transition system [22], 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.

Definition 3.1 *A* probabilistic automaton M *consists of four components:*

(i) *a set* $states(M)$ *of states,*

(ii) *a nonempty set* $start(M) \subseteq states(M)$ *of start states,*

(iii) *an action signature* $sig(M) = (ext(M), int(M))$ *where* $ext(M)$ *and* $int(M)$ *are disjoint sets of* external *and* internal *actions, respectively,*

(iv) *a transition relation* $trans(M) \subseteq states(M) \times acts(M) \times Probs(states(M))$, *where* $acts(M)$ *denotes the set* $ext(M) \cup int(M)$ *of actions.*

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

A probabilistic automaton according to Definition 3.1 is called a *simple probabilistic automaton* in [37]. Observe that an ordinary automaton is a special

case of a probabilistic automaton where each transition leads to a Dirac distribution.

A probabilistic automaton could be seen as a reactive probabilistic transition system of [15]; however, the reactive systems of [15] do not allow the specification of systems with two transitions with that leave from the same state and are labeled by the same action.

3.3 Executions and Probabilistic Executions

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, since each transition leads probabilistically to more than one state. Such a Markov chain is called a *probabilistic execution* and can be visualized as a fully probabilistic automaton.

Formally, we start from the notion of an execution, which is the result of resolving both the nondeterministic and the probabilistic choices in a probabilistic automaton and corresponds to the notion of an execution for ordinary automata.

Definition 3.2 *An* execution fragment 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 $(s_i, a_{i+1}, \mathcal{P}) \in trans(M)$ and $s_{i+1} \in \Omega$. 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 *of M is an execution fragment of M whose first state is a start state of M. Denote by $exec^*(M)$ and $exec(M)$ the sets of finite and all executions of M, respectively.*

An execution α_1 of M is a prefix *of an 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. A probabilistic execution can be seen as the result of unfolding the transition relation of a probabilistic automaton and then choosing a transition from each state.

Definition 3.3 *For each finite execution fragment α and each action a, define a function αa such that $\alpha a(s) = \alpha a s$ for each state s. Recall from Section 3.1*

that the function αa can be lifted to discrete probability spaces.

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

(i) *$states(H) \subseteq exec^*(M)$. Let q range over states of probabilistic executions.*

(ii) *$start(H)$ contains a state of M.*

(iii) *for each transition $(q, a, \mathcal{P})$ of H there is a transition $(lstate(q), a, \mathcal{P}')$ of M such that $\mathcal{P} = qa(\mathcal{P}')$.*

(iv) *each state of H is reachable, where a state q of H is reachable if there is an execution of H whose last state is q.*

A probabilistic execution of M is a probabilistic execution fragment of M whose start state is a start state of M. For each transition $(q, a, \mathcal{P})$ of H, denote $\mathcal{P}$ by $\mathcal{P}_q^H$.

3.4 Events

There is a standard way of defining a probability space $(\Omega_H, \mathcal{F}_H, P_H)$ for a probabilistic execution H. The set Ω_H is the set of limits under prefix of chains of states of H; the σ-field $\mathcal{F}_H$ is the σ-field generated by the set of *cones* $C_\alpha = \{\alpha' \in \Omega_H \mid \alpha \leq \alpha'\}$, where α is a state of H; the measure P_H is the unique measure that extends the measure defined on cones as follows: if $\alpha = s_0 a_1 s_1 \cdots a_n s_n$, then $P_H(C_\alpha) \stackrel{\triangle}{=} P_{q_0}^H[(a_1, q_1)] \cdots P_{q_{n-1}}^H[(a_n, q_n)]$, where each q_i is defined to be $s_0 a_1 s_1 \cdots a_i s_i$.

4 The Idea Behind the Formulation of Coin Lemmas

4.1 Examples of Problems and Applications

We start the illustration of coin lemmas through some examples of problems and algorithms.

Example 4.1 *[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 [27] that there is no symmetric deterministic solution to the dining philosopher's problem, i.e., there is no solution where all philosophers follow the same deterministic algorithm. In [27] 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 and eat. □

Example 4.2 *[Consensus in Exponential Time] There are n processors that propose a value in the set* $\{0, 1\}$. *The values proposed by the processors may be different, but at the end all the processors must agree on the same value, chosen among the values that were proposed. Communication is asynchronous and processors may crash by stopping. This problem is unsolvable [13] since, informally, it is not possible to distinguish between a crashed processor and a slow processor.*

A randomized algorithm that solves the consensus problem works as follows [5]. The algorithm is structured in rounds. At each round the processors interact and try to produce a consistent value to agree on. If the processors cannot agree, then they flip a coin (a different coin for each processor) to choose the value to propose in the next round. With probability $1/2^n$ *all processors choose the same value, and then agreement is possible.* □

Example 4.3 *[Consensus in Polynomial Time] The algorithm described in Example 4.2 works in expected exponential time since there is an exponentially low probability that randomization leads to agreement. In [2] a different way of flipping coins is proposed so that the probability to reach agreement is constant. The processors 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* $\pm Kn$*), then the processors return a value. The value of the shared counter evolves like a stochastic process known as* random walk *[12], and it is possible to use random walk theory to show that there is a constant probability that all processors observe values beyond the same barrier, i.e., all processors return the same value.* □

Example 4.4 *[Choosing a Leader] A randomized algorithm to choose a leader among n processors can be structured in rounds as follows. At each round there are some processors, determined by race conditions, that participate in a game. In the game each processor 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 processors that draw the highest number. If there is a unique winner, then the winner is the leader.*

It is known that if k processors 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 4.1 the process consists of flipping two coins, each one giving a specified value; in Example 4.2 the process consists of flipping n coins, all giving the same value; in Example 4.3 the process consists of computing a random walk where a specific barrier is reached before another specific barrier; in Example 4.4 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 4.4 provides us with several problems. If we consider a computation of the algorithm and we focus on one round, we may observe that some processors, 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 4.4, 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 4.4 does not work. We said that the players of the game are determined by race conditions, 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 processors flip a coin in the randomized dining philosophers algorithm? What happens if not all processors flip a coin in the consensus algorithm of Example 4.2? What happens if none of the barriers is reached in the random walk of Example 4.3 due to the fact that all processors 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 processors disagree and it is sufficient to show that the other processors choose consistent values; in the third case all the processors involved in the coin flipping process crash. The difference between these last three examples and Example 4.4 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 underlying stochastic process is successful are also successful for the algorithm. A well formulated coin lemma ensures that we do not miss any of the dangerous cases where some coins may

not be flipped.

4.2 What is a Coin Lemma

A coin lemma for a specific stochastic process provides us with two objects.

(i) A rule to choose an event in each probabilistic execution of a probabilistic automaton;

(ii) A lower bound on the probability of the events identified by the rule.

Consider again Example 4.1. We represent the algorithm as a probabilistic automaton where each transition corresponds to some action taken by one of the philosophers. We label each transition by the name of the philosopher taking an action and the name of the action being taken. In particular, a transition where philosopher i flips a coin is labeled by flip_i. In order to exploit the main idea behind the algorithm, consider two adjacent philosophers, one of which wants to eat, and consider the process stating that the next coin flip of the two philosophers does not lead to the shared fork. Given a probabilistic execution H of the algorithm, we can check that all the elements of Ω_H where the chosen philosophers flip according to the process above lead to a state where some philosopher eats, thus concluding that the algorithm completes with probability $1/4$. However, the probability in H of the elements of Ω_H that we have just considered is not necessarily $1/4$ since H may contain several executions where one of the chosen philosophers does not flip any coin. A coin lemma, on the other hand, should give us an event associated with H that is guaranteed to have probability at least $1/4$. We could meet the lower bound by considering also all those executions where one of the two philosophers does not flip his coin, but this would be an unnecessarily large event. Alternatively, we can consider those executions where it is possible to fix the values of the unflipped coins so that no philosopher chooses the shared fork (cf. Lemma 5.3). At this point the problem is reduced to checking that all the executions chosen by the coin lemma guarantee the success of the algorithm, a purely nondeterministic problem. The extra executions that appear in the events force us to check explicitly what happens if the stochastic process we have in mind does not take place completely. In particular, in Example 4.1 we would conclude that there are no two neighbor philosophers who want to eat. The reader interested in the full analysis of the Dining Philosophers algorithm using coin lemmas is referred to [28,33].

Example 4.2 can be analyzed with a coin lemma for n coins that are flipped and all give head (cf. Lemma 5.3). In this case the rule for choosing events forces us to consider explicitly those cases where not all processors flip their coin during a round and at the same time all the processors that flip obtain head. In this case we can show, by means of pure nondeterministic analysis, that the processors that do not flip their coin agree already on the value implied by head. Observe that in this specific case a rule where we include all the executions where some processor does not flip any coin would give us

an event that does not imply agreement. The reader interested in the full analysis is referred to [37].

Example 4.3 can be analyzed by using a coin lemma for random walks presented in [34]. The rule considers all those executions where either one barrier is reached or there are finitely many coin flips and no barrier is reached (i.e., there is a way of fixing the results of the unflipped coins so that one barrier is reached).

To analyze Example 4.4 we need a coin lemma for the process of drawing k numbers, each one according to the distribution $P[i] = 1/2^i$, where there is a unique maximum. The rule considers all those executions where either k numbers are drawn and there is a unique maximum, or less than k numbers are drawn (i.e., there is a way of fixing the values of the remaining numbers so that there is a unique maximum). Recall that the lower bound on the probability of the events returned by the rule would not hold if we do not consider the executions where less than k numbers are drawn. Thus, we are forced to analyze explicitly the executions with less than k participants, which are not guaranteed to have a unique maximum, and thus do not guarantee that a leader is elected. At this point, either we can show that there are no executions with less than k participants, or we spot an error in the algorithm.

5 Some Examples of Coin Lemmas

In this section we present some examples of coin lemmas of increasing complexity, where the rule to identify an elementary experiment is based on actions. We emphasize again that actions are not the only way of identifying elementary experiments.

5.1 Simple binary experiment

The first process 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. As an example, if action a identifies the process of flipping a coin, then the set U could be the set of states where the result of the coin flip is head.

Lemma 5.1 *Let M be a 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 $(s, a, \mathcal{P})$ of M, $P[U] \geq p$.*

For each probabilistic execution fragment 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 notational convenience denote $FIRST(a,U)(H)$ by E, and for each state q of H and each set of states X, denote by $\Theta_{q,X}$ the set $\{q' \in \Omega_q^H \mid lstate(q') \in X\}$. Let Θ be the set of states q of H such that action a does not occur in q and action a is the label of the transition leaving from q. Then,

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

By definition of a probabilistic execution, and the hypothesis about p,

$$\sum_{q' \in \Theta_{q,\overline{U}}} P_q^H[q'] \leq 1 - p \tag{2}$$

for each $q \in \Theta$. Thus,

$$P_H[\overline{E}] \leq \sum_{q \in \Theta} P_H[C_q](1-p). \tag{3}$$

Since the cones identified by the elements of Θ are all disjoint, $\sum_{q \in \Theta} P_H[C_q] \leq 1$. Thus,

$$P_H[\overline{E}] \leq (1-p), \tag{4}$$

which is equivalent to $P_H[E] \geq p$. □

Observe from the proof above that the probability of $\overline{E}$ decreases by increasing the probability of scheduling a. The lowest value for the probability of $\overline{E}$ occurs when a is scheduled with probability 1. If we do not include the executions where a is not scheduled in the events returned by the rule $FIRST(a,U)$, then the probability of $\overline{E}$ would increase by decreasing the probability of a, thus violating the lower bound of p for the probability of E.

5.2 First binary experiment among many

The coin lemma of the previous section identifies the binary experiment of interest through the first occurrence of an action a. It is possible to use more complex rules to identify an experiment within a probabilistic execution. In this section we present a coin lemma where the experiment is determined by the first action that occurs among several possible actions.

Lemma 5.2 *Let M be a probabilistic automaton, and let $\mathcal{S}$ be a sequence $(a_1,U_1),\ldots,(a_n,U_n)$ of 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, $P[U] \geq p_i$.*

For each probabilistic execution fragment H of M let $FIRST(\mathcal{S})(H)$ 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.

Then, for each probabilistic execution fragment H of M, $P_H[FIRST(\mathcal{S})(H)] \geq min(p_1, \ldots, p_n)$.

Proof. Let V denote $\{a_1, \ldots, a_n\}$, and let p be the minimum of $\{p_1, \ldots, p_n\}$. For notational convenience, denote $FIRST(\mathcal{S})(H)$ by E, and for each state q of H and each set of states X, denote by $\Theta_{q,X}$ the set $\{q' \in \Omega_q^H \mid lstate(q') \in X\}$. Finally, for each $i \in \{1, \ldots, n\}$, let Θ_i be the set of states q of H such that no action from V occur in q and action a_i is the label of the transition leaving from q. Then,

$$P_H[\overline{E}] = \sum_i \sum_{q \in \Theta_i} \sum_{q' \in \Theta_{q,\overline{U_i}}} P_H[C_q] P_q^H[q']. \tag{5}$$

By definition of a probabilistic execution, and the hypothesis about the p_i's,

$$\sum_{q' \in \Theta_{q,\overline{U_i}}} P_q^H[q'] \leq 1 - p_i \tag{6}$$

for each $i \in \{1, \ldots, n\}$ and each $q \in \Theta_i$. Thus,

$$P_H[\overline{E}] \leq \sum_i \sum_{q \in \Theta_i} P_H[C_q](1 - p_i). \tag{7}$$

By definition of p and a simple algebraic argument,

$$P_H[\overline{E}] \leq (1 - p) \sum_i \sum_{q \in \Theta_i} P_H[C_q]. \tag{8}$$

Since the cones identified by the Θ_i's are all disjoint, $\sum_i \sum_{q \in \Theta_i} P_H[C_q] \leq 1$. Thus,

$$P_H[\overline{E}] \leq (1 - p), \tag{9}$$

which is equivalent to $P_H[E] \geq p$. □

5.3 Beyond first occurrences

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 rule.

Lemma 5.3 *Let M be a probabilistic automaton, and let $\mathcal{S}$ be a sequence $(a_1, U_1), \ldots, (a_n, U_n)$ of 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 $i \in \{1, \ldots, n\}$ and each transition $(s, a_i, \mathcal{P})$ of M, $P[U_i] \geq p_i$.*

For each probabilistic execution fragment H of M let $OCC(k, \mathcal{S})(H)$ be the set of executions α of Ω_H such that either there are less than k occurrences of actions from $\{a_1, \ldots, a_n\}$ in α, or there are at least k occurrences of actions from $\{a_1, \ldots, a_n\}$, and, if a_i is the action that occurs as the k^{th} one, then the state reached after its occurrence is a state of U_i.

Then, for each probabilistic execution fragment H of M,
$P_H[OCC(k, \mathcal{S})(H)] \geq min(p_1, \ldots, p_n)$.

Proof. Similar to the proof of Lemma 5.2 since in that proof the fact that we consider the first occurrence of an action rather than the k^{th} occurrence is not relevant. □

5.4 Conjunctions

It is also possible to consider the conjunction of several properties as well. In order to simplify the notation, we consider only events of the kind $OCC(i, (a, U))$ since, as we have seen in the proof of Lemma 5.2, 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 events 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 5.4 *Let M be a probabilistic automaton, and consider a collection $(k_1, a_1, U_1), \ldots, (k_n, a_n, U_n)$ 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 $\{p_i\}_{i=1,\ldots,n}$ be a collection of real numbers between 0 and 1 such that for each $i \in \{1, \ldots, n\}$ and each transition $(s, a_i, \mathcal{P})$ of M, $P[U_i] \geq p_i$.*

Then, for each probabilistic execution fragment H of M,
$P_H[OCC(k_1, (a_1, U_1))(H) \cap \cdots \cap OCC(k_n, (a_n, U_n))(H)] \geq p_1 \cdots p_n$.

Proof. Simple induction on n. □

5.5 Multiple outcomes

Consider the process of flipping n coins. 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. The coin lemmas seen so far are not adequate.

Similarly, consider the process of rolling n dices. 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?

In this section we describe a coin lemma that can deal with the scenarios outlined above. The underlying stochastic process $\mathcal{P}$ is $\mathcal{P}_1 \times \cdots \times \mathcal{P}_n$, where each Ω_i is the set $\{1, \ldots, k\}$, k a fixed constant, and $\times$ denotes the product of probability spaces. The event E that we consider is any event of $\mathcal{F}$.

We start with the rule. 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. For each execution α of M and each i, $1 \leq i \leq n$, let

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

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

Example 5.5 *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 successful. In the case of the dices E is the set of tuples $(j_1, \ldots, j_n)$ 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 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 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, Lemma 5.6 below states that the probability $P_H[GFIRST(\mathcal{S}, E)(H)]$ is at least p for each H.

Lemma 5.6 *Let $\mathcal{P}$ be $\mathcal{P}_1 \times \cdots \times \mathcal{P}_n$, where each Ω_i is the set $\{1, \ldots, k\}$, k a fixed constant, and let E be an event of $\mathcal{F}$.*

Let M be a 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. Suppose that for each i, j, $1 \leq i \leq n$, $1 \leq j \leq k$, and for each transition $(s, a_i, \mathcal{P})$ of M, $P[U_{i,j}] \geq P_i[j]$.

Then, for each probabilistic execution fragment H of M,
$P_H[GFIRST(\mathcal{S}, E)(H)] \geq P[E]$.

Proof. Simple induction on n. □

It would be desirable to define the most general coin lemma, but this is not possible, since there are several ways of identifying elementary experiments within a probabilistic execution. Furthermore, generality means complexity.

We can generalize the underlying stochastic processes by increasing the number of possible outcomes or by introducing dependencies between elementary experiments. Also we can consider infinite products rather than finite products. Finally, we can define new ways of identifying elementary experiments that are not based simply on occurrences of actions.

In this section we describe a new coin lemma for the process of drawing finitely many natural numbers. We do not assume independence, and we identify elementary experiments in a more general way than a simple count of occurrences of actions at the cost of a more complex notation.

For a sequence x, let x_i denote the i^{th} element of x, and let $x_{<i}$ denote the subsequence of x formed by the first $i-1$ elements of x. 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 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 (no overlap between experiments).

Finally, suppose that for each x, each tuple $(\alpha, a, U_1, U_2, \ldots)$ in A_x, each transition $(lstate(\alpha), a, \mathcal{P}_a)$ 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 x of length m of elements from $N \cup \{*\}$ such that, for each i, if there is a prefix $\alpha' a s$ 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 $x_i = *$. The sequence $x(\alpha)$ contains the outcomes of the elementary experiments that take place. The symbol $*$ means that the corresponding experiment does not take place in α.

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 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 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. Furthermore, coin lemmas force the user into a well-defined probabilistic scenario, drawing his/her attention to the possible interference between probability and nondeterminism, thus reducing the chances of errors due to underestimation of the complexity of the system execution scenario.

In this paper we have identified the key ideas behind the formulation and use of coin lemmas. First of all it is important to be explicit about the stochastic process that is taking place; second, it is important to consider explicitly the possibility that some elementary experiments may not take place: an adversary should never be advantaged by not flipping coins.

Although we have given several examples of coin lemmas in this paper, more coin lemmas will be needed. An example of a coin lemma for random walks can be found in [34]. Furthermore, from the experience that we have gained through several case studies, it seems to be useful to extend the ideas behind coin lemmas to expectations. Once again, the verification example of [34] based on random walks provides an example of the use of coin lemma arguments to derive properties about expectations.

Acknowledgments

I would like to thank the organizers of Probmiv '98 for inviting me to the workshop and the anonymous referees for several useful comments on a draft of this paper.

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 [9], 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] 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.

[6] I. Christoff. Testing equivalences and fully abstract models for probabilistic processes. In Baeten and Klop [4], pages 126–140.

[7] I. Christoff. *Testing Equivalences for Probabilistic Processes.* PhD thesis, Department of Computer Science, Uppsala University, 1990.

[8] 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.

[9] W.R. Cleaveland, editor. *Proceedings of CONCUR 92, Stony Brook, NY, USA*, volume 630 of *Lecture Notes in Computer Science*. Springer-Verlag, 1992.

[10] C. Derman. *Finite State Markovian Decision Processes.* Acedemic Press, 1970.

[11] 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.

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

[13] 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.

[14] 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.

[15] R.J. van Glabbeek, S.A. Smolka, B. Steffen, and C.M.N. Tofts. Reactive, generative, and stratified models of probabilistic processes. In *Proceedings* 5^{th} *Annual Symposium on Logic in Computer Science, Philadelphia, USA*, pages 130–141. IEEE Computer Society Press, 1990.

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

[17] H. Hansson and B. Jonsson. A framework for reasoning about time and reliability. In *Proceedings of the* 10^{th} *IEEE Symposium on Real-Time Systems, Santa Monica, Ca.*, 1989.

[18] H. Hansson and B. Jonsson. A calculus for communicating systems with time and probabilities. In *Proceedings of the* 11^{th} *IEEE Symposium on Real-Time Systems, Orlando, Fl.*, 1990.

[19] B. Jonsson and K.G. Larsen. Specification and refinement of probabilistic processes. In *Proceedings of the 6th IEEE Symposium on Logic in Computer Science*, pages 266–277, Amsterdam, July 1991.

[20] B. Jonsson and W. Yi. Compositional testing preorders for probabilistic processes. In *Proceedings* 10^{th} *Annual Symposium on Logic in Computer Science, San Diego, California*. IEEE Computer Society Press, 1995.

[21] C.C. Jou and S.A. Smolka. Equivalences, congruences, and complete axiomatizations for probabilistic processes. In Baeten and Klop [4], pages 367–383.

[22] R. Keller. Formal verification of parallel programs. *Communications of the ACM*, 7(19):561–572, 1976.

[23] 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.

[24] 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.

[25] K.G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, September 1991.

[26] K.G. Larsen and A. Skou. Compositional verification of probabilistic processes. In Cleaveland [9], pages 456–471.

[27] 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.

[28] 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.

[29] Nancy Lynch and Frits Vaandrager. Forward and backward simulations – Part I: Untimed systems. *Information and Computation*, 121(2):214–233, September 1995.

[30] Nancy Lynch and Frits Vaandrager. Forward and backward simulations – Part II: Timing-based systems. *Information and Computation*, 128(1):1–25, July 1996.

[31] Carroll Morgan and Annabelle McIver. A probabilistic temporal calculus based on expectations. In Lindsay Groves and Steve Reeves, editors, *Proceedings of Formal Methods Pacific '97*. Springer Verlag Singapore, July 1997. http://www.comlab.ox.ac.uk/oucl/groups/probs/bibliography.html.

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

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

[34] 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.

[35] M.O. Rabin. Probabilistic automata. *Information and Control*, 6:230–245, 1963.

[36] 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.

[37] 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.

[38] C. Tofts. A synchronous calculus of relative frequencies. In Baeten and Klop [4].

[39] 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.

[40] S.H. Wu, S. Smolka, and E.W. Stark. Composition and behaviors of probabilistic I/O automata. In B. Jonsson and J. Parrow, editors, *Proceedings of CONCUR 94, Uppsala, Sweden*, volume 836 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994.

[41] W. Yi and K.G. Larsen. Testing probabilistic and nondeterministic processes. In *Protocol Specification, Testing and Verification XII*, pages 47–61, 1992.

[42] L. Zuck. *Past Temporal Logic*. PhD thesis, The Weizman Institute of Science, 1986.