Modeling and Verification of Randomized Distributed Real-Time Systems

by

Roberto Segala

S.M., Electrical Engineering and Computer Science
Massachusetts Institute of Technology
(1992)
Diploma, Computer Science
Scuola Normale Superiore - Pisa
(1991)
Laurea, Computer Science
University of Pisa - Italy
(1991)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1995

Signature of Author_____
Department of Electrical Engineering and Computer Science
May 15, 1995

Certified by_____
Nancy A. Lynch
Professor of Computer Science
Thesis Supervisor

Accepted by_____
Frederic R. Morgenthaler
Chair, Departmental Committee on Graduate Students

Modeling and Verification of Randomized Distributed Real-Time Systems

by

Roberto Segala

Submitted to the Department of Electrical Engineering and Computer Science
on May 15, 1995, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Electrical Engineering and Computer Science

Abstract

Randomization is an exceptional tool for the design of distributed algorithms, sometimes yielding efficient solutions to problems that are inherently complex, or even unsolvable, in the setting of deterministic algorithms. However, this tool has a price: even simple randomized algorithms can be extremely hard to verify and analyze.

This thesis addresses the problem of verification of randomized distributed algorithms. We consider the problem both from the theoretical and the practical perspective. Our theoretical work builds a new mathematical model of randomized distributed computation; our practical work develops techniques to be used for the actual verification of randomized systems. Our analysis involves both untimed and timed systems, so that real-time properties can be investigated.

Our model for randomized distributed computation is an extension of labeled transition systems. A *probabilistic automaton* is a state machine with transitions, where, unlike for labeled transition systems, a transition from a state leads to a discrete probability distribution over pairs consisting of a label and a state, rather than to a single label and a single state. A probabilistic automaton contains *pure nondeterministic behavior* since from each state there can be several transitions, and *probabilistic behavior* since once a transition is chosen the label that occurs and the state that is reached are determined by a probability distribution. The resolution of pure nondeterminism leads to *probabilistic executions*, which are Markov chain like structures. Once the pure nondeterminism is resolved, the probabilistic behavior of a probabilistic automaton can be studied.

The properties of a randomized algorithm are stated in terms of satisfying some other property with a minimal or maximal probability no matter how the nondeterminism is resolved. In stating the properties of an algorithm we also account for the possibility of imposing restrictions on the ways in which the nondeterminism is resolved (e.g., fair scheduling, oblivious scheduling,...). We develop techniques to prove the correctness of some property by reducing the problem to the verification of properties of non-randomized systems. One technique is based on *coin lemmas*, which state lower bounds on the probability that some chosen random draws give some chosen outcomes no matter how the nondeterminism is resolved. We identify a collection of *progress statements* which can be used to prove upper bounds to the expected running time of an algorithm. The methods are applied to prove that the randomized dining philosophers algorithm of Lehmann and Rabin guarantees progress in expected constant time and that the randomized algorithm for agreement of Ben-Or guarantees agreement in expected exponential time.

To ensure that our new model has strong mathematical foundations, we extend some of the

common semantics for labeled transition systems to the probabilistic framework. We define a compositional trace semantics where a trace is replaced by a probability distribution over traces, called a *trace distribution*, and we extend the classical bisimulation and simulation relations in both their strong and weak version. Furthermore, we define *probabilistic forward simulations*, where a state is related to a probability distribution over states. All the simulation relations are shown to be sound for the trace distribution semantics.

In summary, we obtain a framework that accounts for the classical theoretical results of concurrent systems and that at the same time proves to be suitable for the actual verification of randomized distributed real-time systems. This double feature should lead eventually to the easy extension of several verification techniques that are currently available for non-randomized distributed systems, thus rendering the analysis of randomized systems easier and more reliable.

Thesis Supervisor: Nancy A. Lynch
Title: Professor of Computer Science
Keywords: Automata, Distributed Algorithms, Formal Methods, Labeled Transition Systems, Randomized Systems, Real-Time Systems, Verification

Acknowledgements

Eight years ago, when I was getting my high school diploma from the Istituto Tecnico Industriale G. Marconi in Verona, I did not know the meaning of the acronym PhD or even the meaning of the acronym MIT. However, Maurizio Benedetti, my teacher of Computer Science, strongly encouraged me to apply to the Scuola Normale Superiore of Pisa, a place I would have never thought I was qualified for. If it were not for him I probably would not be here writing this acknowledgements section. Also, my first grade teacher, Ines Martini, had an important role in all of this: she is an exceptional person who was able to deal with a terrible kid like me and make him into a person who does not hate school.

Thanks to Rocco De Nicola, my former thesis advisor, for the support that he gave me during my education in Pisa and during my years as a student at MIT; thanks to Sanjoy Mitter who introduced me to MIT and who continuously kept me away from the temptation to focus on just one area of Computer Science.

Nancy Lynch, my advisor here at MIT, deserves strong recognition for the freedom she gave me and for her patience in listening to my fuzzy conjectures, reading and correcting my drafts, improving my English, giving suggestions all over, and most of all, allowing me to benefit from her experience. Whenever I got stuck on something I would invariantly hear her ask "how is it going?", and I was surprised to discover how many times explaining my problems and answering her questions was sufficient to get new ideas.

Albert Meyer was a second advisor for me. Although my research focus is the study of theory for practice, I have always been attracted by nice and clean theoretical results, and Albert was a great source. Several times I stopped by his office, both for research questions or to seek advice on my career choices. He has always been a great source of knowledge and experience, and a great help.

Thanks to Butler Lampson and Albert Meyer for agreeing to be readers of the thesis.

Frits Vaandrager deserves a special thank since he is the person who started me on research. He suggested the topic of my Master's thesis and he guided me during the project although there was an ocean (I was in Italy, he was in the States) between us. It is from my experience with Frits that my idea of studying theory for practice took shape.

The friendly environment here at MIT was a great stimulus for my research. I had many discussions with Rainer Gawlick, Anna Pogosyants, Isaac Saias, and Jørgen Søgaard-Andersen, that lead to some of the papers that I have written in these years. Thanks to all of them. Rainer was also a great advisor for my English and for my understanding of American culture, which sometimes is not so easy to grasp if you are European.

Thanks also go to David Gupta, Alex Russell and Ravi Sundaram for all the help that they gave me on measure theory. Thanks to Mark Tuttle for valuable comments that influenced the presentation of the results of this thesis.

I want to thank several other fellow students and post docs, some of whom are now in better positions, for the help that in various occasions they gave me and for a lot of fun that we had together. In particular, thanks go to Javed Aslam, Margrit Betke, Lenore Cowen, Rosario Gennaro, Shai Halevi, Trevor Jim, Angelika Leeb, Gunter Leeb, Arthur Lent, Victor Luchangco, Daniele Micciancio, Nir Shavit, Mona Singh, Mark Smith, David Wald, H.B. Weinberg. Thanks also to Be Hubbard, our "mum", Joanne Talbot, our group secretary, and Scott Blomquist, our system manager, for their valuable support.

Thanks also go to some other "outsiders" who had an impact on this work. In particular, thanks go to Scott Smolka for useful discussions and for providing me with a rich bibliography on randomized computation, and thanks go to Lenore Zuck for useful discussions on verification techniques.

Last, but not least, a very special thank to my parents, Claudio and Luciana, and to my fiance Arianna for all the love and support that they gave me. This thesis is dedicated to them.

The research in this thesis was supported by NSF under grants CCR-89-15206 and CCR-92-25124, by DARPA under contracts N00014-89-J-1988 and N00014-92-J-4033, and by AFOSR-ONR under contracts N00014-91-J-1046 and F49620-94-1-0199.

Contents

Chapter 1

Introduction

1.1 The Challenge of Randomization

In 1976 Rabin published a paper titled *Probabilistic Algorithms* [Rab76] where he presented efficient algorithms for two well-known problems: *Nearest Neighbors*, a problem in computational geometry, and *Primality Testing*, the problem of determining whether a number is prime. The surprising aspect of Rabin's paper was that the algorithms for Nearest Neighbors and for Primality Testing were efficient, and the key insight was the use of randomized algorithms, i.e., algorithms that can flip fair coins. Rabin's paper was the beginning of a new trend of research aimed at using randomization to improve the complexity of existing algorithms. It is currently conjectured that there are no efficient deterministic algorithms for Nearest Neighbors and Primality Testing.

Another considerable achievement came in 1982, when Rabin [Rab82] proposed a solution to a problem in distributed computing which was known to be unsolvable without randomization. Specifically, Rabin proposed a randomized distributed algorithm for mutual exclusion between n processes that guarantees no-lockout (some process eventually gets to the critical region whenever some process tries to get to the critical region) and uses a test-and-set shared variable with $O(\log n)$ values. On the other hand, Burns, Fisher, Jackson, Lynch and Patterson [BFJ+82] showed that $\Omega(n)$ values are necessary for a deterministic distributed algorithm. Since then, several other randomized distributed algorithms were proposed in the literature, each one breaking impossibility results proved for deterministic distributed algorithms. Several surveys of randomized algorithms are currently available; among those we cite [Kar90, GSB94].

The bottom line is that randomization has proved to be exceptionally useful for problems in distributed computation, and it is slowly making its way into practical applications. However, randomization in distributed computation leaves us with a challenge whose importance increases as the complexity of algorithms increases:

> "*How can we analyze randomized distributed algorithms? In particular, how can we convince ourselves that a randomized distributed algorithm works correctly?*"

The analysis of non-randomized distributed systems is challenging already, due to a phenomenon called *nondeterminism*. Specifically, whenever two systems run concurrently, the relative speeds of the two systems are not known in general, and thus it is not possible to establish a priori the order in which the systems complete their tasks. On the other hand, the ordering of the

completion of different tasks may be fundamental for the global correctness of a system, since, for example, a process that completes a task may prevent another process from completing its task. The structure of the possible evolutions of a system can become intricate quickly, justifying the statement "there is rather a large body of sad experience to indicate that a concurrent program can withstand very careful scrutiny without revealing its errors" [OL82].

The introduction of randomization makes the problem even more challenging since two kinds of nondeterminism arise. We call them *pure nondeterminism* and *probabilistic nondeterminism*. Pure nondeterminism is the nondeterminism due to the relative speeds of different processes; probabilistic nondeterminism is the nondeterminism due to the result of some random draw. Alternatively, we refer to pure nondeterminism as the *nondeterministic behavior* of a system and to probabilistic nondeterminism as the *probabilistic behavior* of a system. The main difficulty with randomized distributed algorithms is that the interplay between probability and nondeterminism can create subtle and unexpected dependencies between probabilistic events; the experience with randomized distributed algorithms shows that "intuition often fails to grasp the full intricacy of the algorithm" [PZ86], and "proofs of correctness for probabilistic distributed systems are extremely slippery" [LR81].

In order to meet the challenge it is necessary to address two main problems.

- **Modeling**: How do we represent a randomized distributed system?
- **Verification**: Given the model, how do we verify the properties of a system?

The main objective of this thesis is to make progress towards answering these two questions.

1.1.1 Modeling

First of all we need a collection of mathematical objects that describe a randomized algorithm and its behavior, i.e., we need a formal model for randomized distributed computation. The model needs to be sufficiently expressive to be able to describe the crucial aspects of randomized distributed computation. Since the interplay between probability and nondeterminism is one of the main sources of problems for the analysis of an algorithm, a first principle guiding our theory is the following:

1. The model should distinguish clearly between probability and nondeterminism.

That is, if either Alice or Bob is allowed to flip a coin, the choice of who is flipping a coin is nondeterministic, while the outcome of the coin flip is probabilistic.

Since the model is to be used for the actual analysis of algorithms, the model should allow the description of randomized systems in a natural way. Thus, our second guiding principle is the following:

2. The model should correspond to our natural intuition of a randomized system.

That is, mathematical elegance is undoubtedly important, but since part of the verification process for an algorithm involves the representation of the algorithm itself within the formal model, the chance of making errors is reduced if the model corresponds closely to our view of a randomized algorithm. A reasonable tradeoff between theory and practice is necessary.

Our main intuition for a computer system, distributed or not, is as a state machine that computes by moving from one state to another state. This intuition leads to the idea of Labeled Transition Systems (LTS) [Kel76, Plo81]. A labeled transition system is a state machine with labels associated with the transitions (the moves from one state to another state). Labeled transition systems have been used successfully for the modeling of ordinary distributed systems [Mil89, Jon91, LV91, LT87, GSSL94], and for their verification [WLL88, SLL93, SGG+93, BPV94]; in this case the labels are used to model communication between several systems. Due to the wide use of labeled transition systems, the extensive collection of verification techniques available, and the way in which labeled transition systems correspond to our intuition of a distributed system, two other guiding principles for the thesis are the following:

3. The new model should extend labeled transition systems.

4. The extension of labeled transition systems should be conservative, i.e., whenever a system does not contain any random choices, our new system should reduce to an ordinary labeled transition system.

In other words our model is an extension of the labeled transition system model so that ordinary non-randomized systems turn out to be a special case of randomized systems. Similarly, all the concepts that we define on randomized systems are generalizations of corresponding concepts of ordinary non-randomized systems. In this way all the techniques available should generalize easily without the need to develop completely new and independent techniques. Throughout the thesis we refer to labeled transition systems as *automata* and to their probabilistic extension as *probabilistic automata*.

1.1.2 Verification

Once the model is built, our primary goal is to use the model to describe the properties that a generic randomized algorithm should satisfy. If the model is well designed, the properties should be easy to state. Then, our second goal is to develop general techniques that can be used for verification.

We investigate verification techniques from two perspectives. On one hand we formalize some of the kinds of the informal arguments that usually appear in existing papers; on the other hand we extend existing abstract verification techniques for labeled transition systems to the probabilistic framework. Examples of abstract techniques include the analysis of traces [Hoa85], which are ordered sequences of labels that can occur during the evolution of a system, and of simulation relations [Mil89, Jon91, LV91], which are relations between the states of two systems such that one system can simulate the transitions of the other via the simulation relation. To provide some intuition for traces and simulations, Figure 1-1 represents three labeled transition systems, denoted by A_1, A_2, and A_3. The empty sequence and the sequences a and ab are the traces of A_1, A_2, and A_3. For example, a computation that leads to ab is the one that starts from s_0, moves to s_1, and then to s_3. The dotted lines from one state to another state (the arrows identify the from-to property) are examples of simulation relations from one automaton to the other. For example, consider the simulation relation from A_3 to A_2. State s_0 of A_3 is related to state s_0 of A_2; states s_1 and s_2 of A_3 are related to state s_1 of A_2; state s_3 of A_3 is related to state s_3 of A_2. The transition of A_3 from s_0 to s_2 with action a is simulated in A_2 by the transition from s_0 to s_1 with label a. There is a strong simulation also from A_2

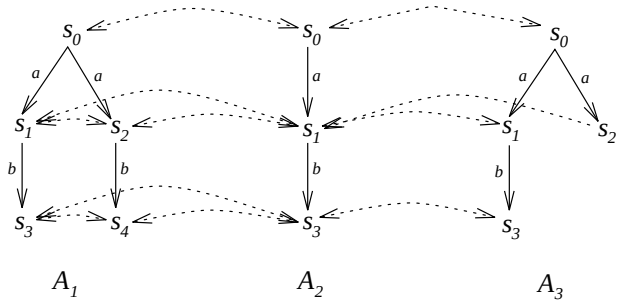


Figure 1-1: Simulation relations for automata.

to A_3 (each state s_i of A_2 is related to state s_i of A_3), from A_1 to A_2, and from A_2 to A_1. There is an even stronger relation between A_1 and A_2, which is called a *bisimulation* and is represented by the double-arrow dotted lines between the states of A_1 and A_2. A bisimulation is an equivalence relation between the states of two automata. In this case each automaton can simulate the transitions of the other via the bisimulation relation.

Direct Verification

In the description of a randomized distributed algorithm pure nondeterminism represents the undetermined part of its behavior, namely, in what order the processes are scheduled. Scheduling processes is the activity of removing the nondeterminism, and the object that does the scheduling is usually referred to as a *scheduler* or an *adversary*. The intuition behind the name "adversary" is in proving the correctness of an algorithm a scheduler is viewed as a malicious entity that degrades the performance of the system as much as possible.

Once the nondeterminism is removed, a system looks like a Markov chain, and thus it is possible to reason about probabilities. A common argument is then

> "*no matter how the scheduler acts, the probability that some good property holds is at least p.*"

Actually, in most of the existing work p is 1, since the proofs are easier to carry out in this case. In this thesis we are interested in every p since we are concerned also with the time complexity of an algorithm. Throughout the thesis it will become clear why we need every p for the study of time complexity.

One of our major goals is to remove from the informal arguments of correctness all "dangerous" statements, i.e., all statements that rely solely on intuition rather than on actual deductions, and yet keep the structure of a proof simple. In other words, we want to provide tools that allow people to argue as before with a significantly higher confidence that what they say is correct. Then, we want to develop techniques that allow us to decompose the verification task of complex properties into simpler verification tasks. This feature is important for scalability. Here we give examples of two issues that we believe to be important.

- *Make sure that you know what probability space you are working in.* Or, at least, make sure that you are working in a probability space. This is a rule of thumb that is valid in other fields like Information Theory and Detection Theory. Probability is very tricky. The

fact that a specific probability space was not identified was the reason for a bug discovered by Saias [Sai92] in the original algorithm of Rabin [Rab82], later fixed by Kushilevitz and Rabin [KR92]. Of course, in order to make sure we know what probability spaces we are working in, we need some easy mechanisms to identify those probability spaces. Such mechanisms were not available in 1982.

- *Avoid arguments of the kind "now the worst thing that can happen is the following."* These arguments are usually based on the intuition that the designers have about their own algorithm. Specifically, as has happened in the past, the designers argue based on worst cases they can think of rather than the actual worst case. What is missing is a proof showing that the worst case has been identified. A much better statement would be "no matter what happens, something else will happen", since it does not require us to identify the worst scenario. Using our methodology, Aggarwal [Agg94] discovered a bug in an algorithm designed by himself and Kutten [AK93] which was due to an argument of the kind cited above. Similarly, we discovered a bug in the timing analysis of the mutual exclusion algorithm of Pnueli and Zuck [PZ86]. This bug arose for the same reason.

The reader familiar with existing work, and in particular familiar with model checking, may be a bit puzzled at this point. There is a considerable amount of work on model checking of randomized distributed systems, and yet we are introducing new techniques. Furthermore, although there is some ongoing work on automating part of the proof methods developed in this thesis [PS95], we do not address any decidability issue here. Our favorite analogy to justify our approach is that we view model checking as the program "Mathematica", a popular program for symbolic manipulation of analytic expressions. If we are given a simple analytical problem, we can use Mathematica to get the solution from a computer. On the other hand, if we have a complex analytical problem, say a complex function that we have defined, and we want to verify that it respects some specific constraints, or maybe we want to find the constraints, then things are very different, since the problem in general is undecidable, i.e., not solvable by a computer. We can plot part of the given function using Mathematica and have a rough idea of whether it satisfies the desired constraints. If the plot shows that the function violates some of the constraints, then we have to change either the function or the constraints; if the plot shows that the function does not violate the constraints, then we can start to use all the tools of *analysis* to prove that the given function satisfies the constraints. In this way Mathematica saves us a lot of time. In using the analytical tools we need to use our creativity and our intuition about the problem so that we can solve its undecidable part. We view our research as building the analytical tools.

Simulations

The study of traces and simulations carried out in the thesis contributes more directly to theory than to practice. In particular, we do not give any examples of verification using simulations. However, due to the success that simulation relations have had for the verification of ordinary labeled transition systems, it is likely that the same methods will also work for randomized systems.

A considerable amount of research has been carried out in extending trace semantics and simulation relations to the probabilistic case, especially within process algebras [Hoa85, Mil89,

BW90]; however, most of the existing literature does not address pure nondeterminism, and thus it has limited practical applicability. We believe it is important to have a model that is both useful for realistic problems and accounts for the existing theoretical work. In particular, based on some of the interpretations that are given to nondeterminism within ordinary automata, we realize that, also in the probabilistic case, pure nondeterminism can be used to express much more than just the relative speeds of processes running concurrently. Specifically, nondeterminism can be used to model the following phenomena.

1. *Scheduling freedom.* This is the classical use of nondeterminism, where several processes run in parallel and there is freedom in the choice of which process performs the next transition.

2. *External environment.* Some of the labels can represent communication events due to the action of some external user, or more generally, to the action of an *external environment*. In this case nondeterminism models the arbitrary behavior of the external environment, which is chosen by an adversary.

3. *Implementation Freedom.* A probabilistic automaton is viewed as a specification, and nondeterminism represents implementation freedom. That is, if from some state there are two transitions that can be chosen nondeterministically, then an implementation can have just one of the two transitions. In this case an adversary chooses the implementation that is used.

It is important to recognize that, in the labeled transition system model, the three uses of nondeterminism described above can coexist within the same automaton. It is the specific interpretation that is given to the labels that determines what is expressed by nondeterminism at each point.

1.2 Organization of the Thesis

The thesis is divided in two main parts: the first part deals with the untimed model and the second part deals with the timed model. The second part relies heavily on the first part and adds a collection of results that are specific to the analysis of real-time properties. We describe the technical contributions of the thesis chapter by chapter.

An Overview of Related Work. Chapter 2 gives an extensive overview of existing work on modeling and verification of randomized distributed systems.

Preliminaries. Chapter 3 gives the basics of probability theory that are necessary to understand the thesis and gives an overview of the labeled transition systems model. All the topics covered are standard, but some of the notation is specific to this thesis.

Probabilistic Automata. Chapter 4 presents the basic probabilistic model. A *probabilistic automaton* is a state machine whose transitions lead to a probability distribution over the labels that can occur and the new state that is reached. Thus, a transition describes the probabilistic behavior of a probabilistic automaton, while the choice of which transition to perform describes

the nondeterministic behavior of a probabilistic automaton. A computation of a probabilistic automaton, called a *probabilistic execution*, is the result of resolving the nondeterminism in a probabilistic automaton, i.e., the result of choosing a transition, possibly using randomization, from every point. A probabilistic execution is described essentially by an infinite tree with probabilities associated with its edges. On such a tree it is possible to define a probability space, which is the object through which the probabilistic properties of the computation can be studied. We extend the notions of finiteness, prefix and suffix of ordinary executions to the probabilistic framework and we extend the parallel composition operator. Finally, we show how to project a probabilistic execution of a compound probabilistic automaton onto one of its components and we show that the result is a probabilistic execution of the component. Essentially, we show that the properties of ordinary automata are preserved in the probabilistic framework. The probabilistic model is an extension of ordinary automata since an ordinary automaton can be viewed as a probabilistic automaton where each transition leads just to one action and one state.

Direct Verification: Stating a Property. Chapter 5 shows how to formalize commonly used statements about randomized algorithms and shows how such formal statements can be manipulated. We start by formalizing the idea of an *adversary*, i.e., the entity that resolves the nondeterminism of a system in a malicious way. An adversary is a function that, given the past history of a system, chooses the next transition to be scheduled, possibly using randomization. The result of the interaction between an adversary and a probabilistic automaton is a probabilistic execution, on which it is possible to study probabilistic properties. Thus, given a collection of adversaries and a specific property, it is possible to establish a bound on the probability that the given property is satisfied under any of the given adversaries. We call such bound statements *probabilistic statements*. We show how probabilistic statements can be combined together to yield more complex statements, thus allowing for some form of compositional verification. We introduce a special kind of probabilistic statement, called a *progress statement*, which is a probabilistic extension of the *leads-to* operator of UNITY [CM88]. Informally, a progress statement says that if a system is started from some state in a set of states U, then, no matter what adversary is used, a state in some other set of states U' is reached with some minimum probability p. Progress statements can be combined together under some general conditions on the class of adversaries that can be used.

Finally, we investigate the relationship between deterministic adversaries (i.e., adversaries that cannot use randomness in their choices) and general adversaries. We show that for a large class of collections of adversaries and for a large class of properties it is sufficient to analyze only deterministic adversaries in order to derive statements that concern general adversaries. This result is useful in simplifying the analysis of a randomized algorithm.

Direct Verification: Proving a Property. Chapter 6 shows how to prove the validity of a probabilistic statement from scratch. We introduce a collection of *coin lemmas*, which capture a common informal argument on probabilistic algorithms. Specifically, for many proofs in the literature the intuition behind the correctness of an algorithm is based on the following fact: if some specific random draws give some specific results, then the algorithm guarantees success. Then, the problem is reduced to showing that, no matter what the adversary does, the specific random draws give the specific results with some minimum probability. The coin

lemmas can be used to show that the specific random draws satisfy the minimum probability requirement; then, the problem is reduced to verifying properties of a system that does not contain probability at all. Factoring out the probability from a problem helps considerably in removing errors due to unexpected dependencies.

We illustrate the method by verifying the correctness of the randomized dining philosophers algorithm of Lehmann and Rabin [LR81] and the algorithm for randomized agreement with stopping faults of Ben-Or [BO83]. In both cases the correctness proof is carried out by proving a collection of progress statements using some coin lemmas.

Finally, we suggest another technique, called the *partition technique*, that departs considerably from the coin lemmas and that appears to be useful in some cases. We illustrate the partition technique on a toy resource allocation protocol, which is one of the guiding examples throughout Chapters 5 and 6.

Hierarchical Verification: Trace Distributions. Chapter 7 extends the trace-based semantics of ordinary automata [Hoa85] to the probabilistic framework. A trace is a ordered sequence of labels that occur in an execution; a *trace distribution* is the probability distribution on traces induced by a probabilistic execution. We extend the trace preorder of ordinary automata (inclusion of traces) to the probabilistic framework by defining the *trace distribution preorder*. However, the trace distribution preorder is not preserved by the parallel composition operator, i.e., it is not a *precongruence*. Thus, we define the *trace distribution precongruence* as the coarsest precongruence that is contained in the trace distribution preorder. Finally, we show that there is an elementary probabilistic automaton called the *principal context* that distinguishes all the probabilistic automata that are not in the trace distribution precongruence relation. This leads us to an alternative characterization of the trace distribution precongruence as inclusion of *principal trace distributions*.

Hierarchical Verification: Simulations. Chapter 8 extends the verification method based on simulation relations to the probabilistic framework. Informally, a simulation relation from one automaton to another automaton is a relation between the states of the two automata that allows us to embed the transition relation of one automaton in the other automaton. In the probabilistic framework a simulation relation is still a relation between states; however, since a transition leads to a probability distribution over states, in order to say that a simulation relation embeds the transition relation of a probabilistic automaton into another probabilistic automaton we need to extend a relation defined over states to a relation defined over probability distributions over states. We generalize the strong and weak bisimulation and simulation relations of Milner, Jonsson, Lynch and Vaandrager [Mil89, Jon91, LV91] to the probabilistic framework. Then, we introduce a coarser simulation relation, called a *probabilistic forward simulation*, where a state is related to a probability distribution over states rather than to a single state. We prove an *execution correspondence theorem* which, given a simulation relation from one probabilistic automaton to another probabilistic automaton, establishes a strong correspondence between each probabilistic execution of the first probabilistic automaton and one of the probabilistic executions of the second automaton. Based on the execution correspondence theorem, we show that each of the relations presented in the chapter is sound for the trace distribution precongruence. Thus, simulation relations can be used as a sound technique to prove principal trace distribution inclusion.

Probabilistic Timed Automata. Chapter 9 starts the second part of the thesis. We extend probabilistic automata with time following the approach of Lynch and Vaandrager [LV95], where passage of time is modeled by means of transitions labeled with positive real numbers. In order to use most of the untimed theory, we force time-passage transition not to be probabilistic. We extend probabilistic executions to the timed framework, leading to probabilistic timed executions, and we show the relationship between probabilistic executions and probabilistic timed executions. The main idea is that in several circumstances it is sufficient to analyze the probabilistic executions of a system in order to study its real-time behavior.

Direct Verification: Time Complexity. Chapter 10 introduces new techniques for the verification of real-time properties of a randomized algorithm. The techniques of Chapter 5 still apply; however, due to the presence of time, it is possible to study the time complexity of an algorithm. We augment the progress statements of Chapter 5 with an upper bound t to state the following: if a system is started from some state in a set of states U, then, no matter what adversary is used, a state of some other set of states U' is reached within time t with some minimum probability p. Based on these *timed progress statements*, we show how to derive upper bounds on the expected time to reach some set of states. We illustrate the technique by showing that the randomized dining philosophers algorithm of Lehmann and Rabin [LR81] guarantees progress within expected constant time.

By extending the technique for the analysis of expected time, we show how to derive bounds on more abstract notions of complexity. In particular, we consider the algorithm for randomized agreement of Ben-Or as an example. The algorithm of Ben-Or runs in stages. From the way the algorithm is structured, it is not possible to give meaningful bounds on the time it takes to make progress from any reachable state. However, using abstract complexities, it is easy to prove an upper bound on the expected number of stages that are necessary before reaching agreement. Once an upper bound on the expected number of stages is derived, it is easy to derive an upper bound on the expected time to reach agreement.

Hierarchical Verification: Timed Trace Distributions and Timed Simulations. Chapters 11 and 12 extend the trace distribution precongruence and the simulation relations of the untimed framework to the timed framework. A trace is replaced by a *timed trace*, where a timed trace is a sequence of labels paired with their time of occurrence plus a limit time. The timed trace distribution precongruence is characterized by a *timed principal context*, which is the principal context augmented with arbitrary time-passage transitions. All the timed simulation relations are shown to be sound for the timed trace distribution precongruence. All the results are proved by reducing the problem to the untimed framework.

Conclusion. Chapter 13 gives some concluding remarks and several suggestions for further work. Although this thesis builds a model for randomized computation and shows that it is sufficiently powerful for the analysis of randomized distributed real-time algorithms, it just discovers the tip of the iceberg. We propose a methodology for the analysis of randomization, and we give several examples of the application of such methodology; however, there are several other ways to apply our methodology. It is very likely that new probabilistic statements, new results to combine probabilistic statements, and new coin lemmas can be developed based on the study of other algorithms; similarly, the fundamental idea behind the trace semantics that we

present can be used also for other kinds of observational semantics like failures [Hoa85, DH84]. We give hints on how it is possible to handle liveness within our model and state what we know already. Furthermore, we give ideas of what is possible within restricted models where some form of I/O distinction like in the work of Lynch and Tuttle [LT87] or some timing restriction like in the work of Merritt, Modugno and Tuttle [MMT91] is imposed. Finally, we address the issue of relaxing some of the restrictions that we impose on the timed model.

1.3 Reading the Thesis

The two parts of the thesis, the untimed and the timed part, proceed in parallel: each chapter of the untimed part is a prerequisite for the corresponding chapter in the timed part. Each part is subdivided further into two parts: the direct verification and the hierarchical verification. The two parts can be read almost independently, although some knowledge of the direct verification method can be of help in reading the hierarchical method. The direct method is focused mainly on verification of algorithms, while the hierarchical method is focused mainly on the theoretical aspects of the problem. Further research should show how the hierarchical method can be of significant help for the analysis of randomized algorithms.

Each chapter starts with an introductory section that gives the main motivations and an overview of the content of the chapter. Usually, the more technical discussion is concentrated at the end. The same structure is used for each section: the main result and short proofs are at the beginning of each section, while the long proofs and the more technical details are given at the end. A reader can skip the proofs and the most technical details on a first reading in order to have a better global picture. It is also possible to read just Chapter 3 and the first section (including subsections) of Chapters 4 to 12, and have a global view of the results of the thesis. In a second reading, the interested reader can concentrate on the proofs and on the technical definitions that are necessary for the proofs. The reader should keep in mind that several proofs in the thesis are based on similar techniques. Such techniques are explained in full detail only the first time they are used.

A reader interested only in the techniques for the direct verification of algorithms and not interested in the arguments that show the foundations of the model can avoid reading the proofs. Moreover, such a reader can just glance over Section 4.2.6, and skip Sections 4.2.7, 4.3, and 4.4. In the timed framework the reader interested just in the techniques for the direct verification of algorithms can skip all the comparison between the different types of probabilistic timed executions and concentrate more on the intuition behind the definition of a probabilistic timed execution.

Chapter 2

An Overview of Related Work

In this chapter we give an extensive overview of existing work on modeling and verification of randomized distributed systems. We defer the comparison of our work with the existing work to the end of each chapter. Some of the descriptions include technical terminology which may be difficult to understand for a reader not familiar with concurrency theory. Such a reader should focus mainly on the high level ideas and not worry about the technical details. The rest of the thesis presents our research without assuming any knowledge of concurrency theory. We advise the reader not familiar with concurrency theory to read this chapter again after reading the thesis.

There have been two main research directions in the field of randomized distributed real-time systems: one focused mainly on modeling issues using process algebras [Hoa85, Mil89, BW90] and labeled transition systems [Kel76, Plo81] as the basic mathematical objects; the other focused mainly on verification using Markov chains as the basic model and temporal logic arguments [Pnu82] and model checking [EC82, CES83] as the basic verification technique. Most of the results of the first of the research directions fail to model pure nondeterminism, while the results of the second of the research directions model pure nondeterminism successfully, but not in its full generality. As expressed at the end of Section 1.1.2, pure nondeterminism arises only in the choice of what process is performing the next instruction at each moment. Below we summarize the results achieved in both of the research directions. Furthermore, at the end of each chapter we add a section where we explain how the results described in this section are related to our research.

2.1 Reactive, Generative and Stratified Models

We present some of the existing work on modeling which is based on a classification due to van Glabbeek, Smolka, Steffen and Tofts [GSST90]. They define three types of processes: *reactive*, *generative*, and *stratified*.

- *Reactive model:* Reactive processes consist of states and labeled transitions associated with probabilities. The restriction imposed on a reactive process is that for each state the sum of the probabilities of the transitions with the same label is 1.
- *Generative model:* Generative processes consist of states and labeled transitions associated with probabilities. The restriction imposed on a generative process is that for each state

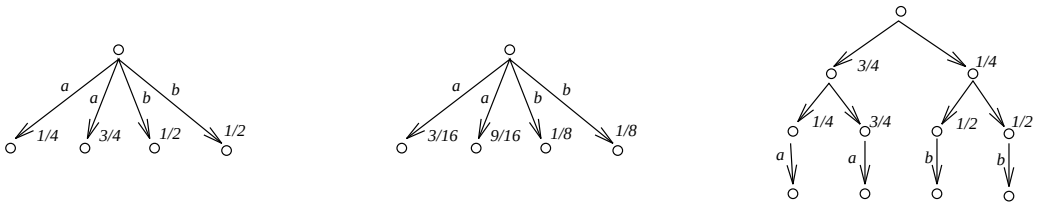


Figure 2-1: Reactive, generative and stratified processes, from left to right.

either there are no outgoing transitions, or the sum of the probabilities of all the outgoing transitions is 1.

- *Stratified model:* Stratified processes consist of states, unlabeled transitions associated with probabilities, and labeled transitions. The restriction imposed on a stratified process is that for each state either there is exactly one outgoing labeled transition, or all the outgoing transitions are unlabeled and the sum of their probabilities is 1.

Figure 2-1 gives an example of a reactive, a generative, and a stratified process. Informally, reactive processes specify for each label (also called action) the probability of reaching other states; generative processes also give additional information concerning the relative probabilities of the different actions; stratified processes add some probabilistic structure to generative processes. Observe that among the three models above only the reactive model has a structure that can be used to express some form of pure nondeterminism (what action to perform), although in van Glabbeek et al. [GSST90] this issue is not considered.

2.1.1 Reactive Model

Rabin [Rab63] studies the theory of probabilistic automata, which are an instance of the reactive model. He defines a notion of a language accepted by a probabilistic automaton relative to a cut point λ and shows that there are finite state probabilistic automata that define non-regular languages.

Larsen and Skou [LS89, LS91] define a bisimulation type semantics, called *probabilistic bisimulation*, and a logic, called *probabilistic model logic* (PML), for reactive processes, and they introduce a notion of testing based on sequential tests and a copying facility. They show that two processes that satisfy the *minimal probability assumption* are probabilistically bisimilar if and only if they satisfy exactly the same PML formulas, and that two processes that satisfy the minimal probability assumption and that are not probabilistically bisimilar can be distinguished through testing with a probability arbitrarily close to 1. The minimum probability assumption states that for every state the probability of each transition is either 0 or is above some minimal value. This condition corresponds to the image-finiteness condition for non-probabilistic processes. Bloom and Meyer [BM89] relate the notions of probabilistic and non-probabilistic bisimilarity by showing that two non-probabilistic finitely branching processes P and Q are bisimilar if and only if there exists an assignment of probabilities to the transitions of P and Q such that the corresponding reactive processes P' and Q' are probabilistically bisimilar.

Larsen and Skou [LS92] introduce a synchronous calculus for reactive processes where the probabilistic behavior is obtained through a binary choice operator parameterized by a prob-

ability p. They define a bisimulation relation on the new calculus, and they introduce a new extended probabilistic logic (EPL) which extends PML in order to support decomposition with respect to parallel composition. Both the probabilistic bisimulation and the extended probabilistic logic are axiomatized.

2.1.2 Generative and Stratified Models

Giacalone, Jou and Smolka [GJS90] define a process algebra for generative processes, called PCCS, which can be seen as a probabilistic extension of Milner's SCCS [Mil93]. In PCCS two processes synchronize at every transition regardless of the action that they perform. That is, if one process performs a transition labeled with action a with probability p_a and another process performs a transition labeled with b with probability p_b, then the two processes together can perform a transition labeled with ab with probability $p_a p_b$. The authors provide an equational theory for PCCS based on the probabilistic bisimulation of Larsen and Skou [LS89], and provide an axiomatization for probabilistic bisimulation (the axiomatization is shown to be sound and complete in [JS90]). Furthermore, the authors define a notion of ϵ-bisimulation, where two processes can simulate each other's transition with a probability difference at most ϵ. Based on ϵ-bisimulation, the authors define a metric on generative processes.

Jou and Smolka [JS90] define trace and failure equivalence for generative processes. They show that, unlike for nondeterministic transition systems, maximality of traces and failures does not increase the distinguishing power of trace and failure equivalence, where by maximality of a trace we mean the probability to produce a specific trace and then terminate. More precisely, knowing the probability of each finite trace of a generative process gives enough information to determine the probability that a finite trace occurs leading to termination; similarly, knowing the probability of every failure of a generative process gives enough information to determine the probability of each maximal failure. Jou and Smolka show also that the trace and failure equivalences are not congruences. Our probabilistic executions are essentially generative proceses, and our trace distributions are essentially the trace semantics of Jou and Smolka. In our case the properties shown by Jou and Smolka follow directly from measure theory.

Van Glabbeek et al. [GSST90] state that the generative model is more general than the reactive model in the sense that generative processes, in addition to the relative probabilities of transitions with the same label, contain information about the relative probabilities of transitions with different labels. They show also that the stratified model is a generalization of the generative model in the sense that a probabilistic choice in the generative model is refined by a structure of probabilistic choices in the stratified model. Formally, the authors give three operational semantics to PCCS, one reactive, one generative, and one stratified, and show how to project a stratified process into a generative process and how to project a generative process into a reactive process, so that the operational semantics of PCCS commute with the projections. The reactive and generative processes of Figure 2-1 are the result of the projection of the generative and stratified processes, respectively, of Figure 2-1. Finally, the authors define probabilistic bisimulation for the generative and for the stratified models and show that bisimulation is a congruence in all the models and that bisimulation is preserved under projection from one model to the other. The results of van Glabbeek et al. [GSST90], however, are based on the fact that parallel composition is synchronous.

Tofts [Tof90] introduces a *weighted synchronous calculus* whose operational semantics resem-

bles the stratified model. The main difference is that the weights associated with the transitions are not probabilities, but rather *frequencies*, and thus their sums are not required to be 1. Tofts defines two bisimulation relations that are shown to be congruences. The first relation is sensitive to the actual frequencies of the transitions leaving from a state, while the second relation is sensitive only to the relative frequencies of the transitions leaving from a state. In particular, the second relation coincides with the stratified bisimulation of van Glabbeek et al. [GSST90] after normalizing to 1 the frequencies of the transitions that leave from every state. The advantage of Tofts' calculus is that it is not necessary to restrict the syntax of the expressions so that the weights of the choices at any point sum to 1 (such a restriction is imposed in PCCS). Moreover, it is possible to define a special weight ω that expresses *infinite frequency* and can be used to express priorities. A similar idea to express priorities is used by Smolka and Steffen in [SS90], where the stratified semantics of PCCS is extended with 0-probability transitions.

Baeten, Bergstra and Smolka [BBS92] define an algebra, $pr\mathrm{ACP}_I^-$, which is an extension of ACP [BW90] with generative probabilities. The authors show that $pr\mathrm{ACP}_I^-$ and a weaker version of ACP (ACP_I^-) are correlated in the sense that ACP_I^- is the homomorphic image of $pr\mathrm{ACP}_I^-$ in which the probabilities are forgotten. The authors also provide a sound and complete axiomatization of probabilistic bisimulation.

Wu, Smolka and Stark [WSS94] augment the I/O automaton model of Lynch and Tuttle [LT87] with probability and they study a compositional behavioral semantics which is also shown to be fully abstract with respect to probabilistic testing. A test is a probabilistic I/O automaton with a success action w. The model is reactive for the input actions and generative for the output actions. This allows the authors to define a meaningful parallel composition operator, where two probabilistic I/O automata synchronize on their common actions and evolve independently on the others. In order to deal with the nondeterminism that arises from parallel composition, the authors attach a *delay parameter* to each state of a probabilistic I/O automaton, which can be seen as the parameter of an exponential probability distribution on the time of occurrence of the next *local* (i.e., output or internal) action. Whenever there is a conflict for the occurrence of two local actions of different probabilistic I/O automata, the delay parameters associated with the states are used to determine the probability with which each action occurs. The behavior of a probabilistic I/O automaton A is a function $\mathcal{E}^A$ that associates a functional $\mathcal{E}^A_\beta$ with each finite trace β . If the length of β is n, then $\mathcal{E}^A_\beta$ takes a function f that given $n+1$ delay parameters computes an actual delay, and returns the expected value of f applied to the delay parameters of the computations of A that lead to β.

2.2 Models based on Testing

Research on modeling has also focused on extending the testing preorders of De Nicola and Hennessy [DH84] to probabilistic processes. To define a testing preorder it is necessary to define a notion of a *test* and of how a test interacts with a process. The interaction between a test and a process may lead to *success* or *failure*. Then, based on the success or failure of the interactions between a process and a test, a preorder relation between processes is defined. Informally, a test checks whether a process has some specific features: if the interaction between a test and a process is successful, then the process has the desired feature.

Ivan Christoff [Chr90b, Chr90a] analyzes generative processes by means of testing. A test is a nondeterministic finite-state process, and the interaction between a process and a test is

obtained by performing only those actions that both the processes offer and by keeping the relative probability of each transition unchanged. Four testing preorders are defined, each one based on the probability of the traces of the interaction between a process and a test. Christoff also provides a fully abstract denotational semantics for each one of the testing preorders: each process is denoted by a mapping that given an *offering* and a trace returns a probability. An offering is a finite sequence of non-empty sets of actions, and, informally, describes the actions that the environment offers to a process during the interaction between the process and a test.

Linda Christoff [Chr93] builds on the work of Ivan Christoff and defines three linear semantics for generative processes: the *trace* semantics, the *broom* semantics, and the *barbed* semantics. The relations are defined in a style similar to the denotational models of Ivan Christoff, and, in particular, the trace and barbed semantics coincide with two of the semantics of [Chr90b]. Linda Christoff also defines three linear-time temporal logics that characterize her three semantics and provides efficient model checking algorithms for the recursion-free version of the logics.

Testing preorders that are more in the style of De Nicola and Hennessy [DH84] are presented by Yi and Larsen in [YL92], where they define a process algebra with all the operators of CCS plus a binary probabilistic choice operator parameterized by a probability p. Thus, the calculus of Yi and Larsen allows for nondeterminism. A test is a process of their calculus with an additional label w. Depending on how the nondeterminism is resolved, w occurs with different probabilities in the interaction between a process and a test. Then, Yi and Larsen define a *may* preorder, which is based on the highest probability of occurrence of w, and a *must* preorder, which is based on the lowest probability of occurrence of w. The two preorders are shown to coincide with the testing preorders of De Nicola and Hennessy [DH84] when no probability is present. In more recent work Jonsson, Ho-Stuart and Yi [JHY94] give a characterization of the may preorder based on tests that are not probabilistic, while Jonsson and Yi [JY95] give a characterization of the may and must preorders based on general tests.

Cleaveland, Smolka and Zwarico [CSZ92] introduce a testing preorder on reactive processes. A test is a reactive process with a collection of *successful* states and a *non-observable* action. The interaction between a test and a process allows an observable action to occur only if the two processes allow it to occur, and allows the non-observable action to occur if the test allows it to occur. The result is a generative process, where each of the actions that occur is chosen according to a uniform distribution (thus the formalism works only for finitely many actions). Two processes are compared based on the probability of reaching a successful state in the interaction between a process and a test. The authors show that their testing preorder is closely connected to the testing preorders of De Nicola and Hennessy [DH84] in the sense that if a process passes a test with some non-zero probability, then the non-probabilistic version of the process (the result of removing the probabilities from the transition relation of the process) *may* pass the non-probabilistic version of the test, and if a process passes a test with probability 1, then the non-probabilistic version of the process *must* pass the non-probabilistic version of the test. An alternative characterization of the testing preorder of Cleaveland et al. [CSZ92] is provided by Yuen, Cleaveland, Dayar and Smolka [YCDS94]. A process is represented as a mapping from *probabilistic traces* to $[0, 1]$, where a probabilistic trace is an alternating sequence of actions and probability distributions over actions. Yuen et al. use the alternative characterization to show that the testing preorder of Cleaveland et al. [CSZ92] is an equivalence relation.

2.3 Models with Nondeterminism and Denotational Models

2.3.1 Transitions with Sets of Probabilities

Jonsson and Larsen [JL91] introduce a new kind of probabilistic transition system where the transitions are labeled by sets of allowed probabilities. The idea is to model specifications where the probabilities associated with the transitions are not completely specified. They extend the bisimulation of Larsen and Skou [LS89] to the new framework and they propose two criteria for refinement between specifications. One criterion is analogous to the definition of simulations between non-probabilistic processes; the other criterion is weaker and regards a specification as a set of probabilistic processes. Refinement is then defined as inclusion of probabilistic processes. Finally, Jonsson and Larsen present a complete method for verifying containment between specifications.

2.3.2 Alternating Models

Hansson and Jonsson [HJ89, HJ90] develop a probabilistic process algebra based on an *alternating model*. The model of Hansson and Jonsson, which is derived from the Concurrent Markov Chains of Vardi [Var85], is a model in which there are two kinds of states: *probabilistic states*, whose outgoing transitions are unlabeled and lead to nondeterministic states, and *nondeterministic states*, whose outgoing transitions are labeled and lead to probabilistic states. Only the transitions leaving from probabilistic states are probabilistic, and for each probabilistic state the probabilities of the outgoing transitions add to 1. The authors define a strong bisimulation semantics in the style of Larsen and Skou [LS89] for which they provide a sound and complete axiomatization. The model of Hansson and Jonsson [HJ90] differs substantially from the models of van Glabbeek et al. [GSST90] in that there is a clear distinction between pure nondeterminism and probability. The model could be viewed as an instance of the reactive model; however, the parallel composition operation defined by Hansson and Jonsson [HJ90] is asynchronous, while the classification of van Glabbeek et al. [GSST90] works only for synchronous composition. A complete presentation of the work of Hansson and Jonsson [HJ89, HJ90] appears in Hansson's PhD thesis [Han91], later published as a book [Han94]. Our simple probabilistic automata are very similar in style to the objects of Hansson's book.

2.3.3 Denotational Semantics

Seidel [Sei92] extends CSP [Hoa85] with probability. The extension is carried out in two steps. In the first step a process is a probability distribution over traces; in the second step, in order to account for the nondeterministic behavior of the environment, a process is a conditional probability measure, i.e., an object that given a trace, which is meant to be produced by the external environment, returns a probability distribution over traces.

Jones and Plotkin [JP89] use a category theoretic approach to define a probabilistic powerdomain, and they use it to give a semantics to a language with probabilistic concurrency. It is not known yet how the semantics of Jones and Plotkin compares to existing operational semantics.

2.4 Models with Real Time

There are basically two models that address real time issues. One model is the model of Hansson and Jonsson [Han94], where special χ actions can appear in the transitions. The occurrence of an action χ means that time has elapsed, and the amount of time that elapses in a computation is given by the number of occurrences of action χ. Thus, the time domain of Hansson and Jonsson's model is discrete.

The other model is based on *stochastic process algebras* and is used in the field of performance analysis. In particular, actions are associated with durations, and the durations are expressed by random variables. In order to simplify the analysis, the random variables are assumed to have an exponential probability distribution, which is memoryless. Research in this area includes work from Götz, Herzog and Rettelbach [GHR93], from Hillston [Hil94], and from Bernardo, Donatiello and Gorrieri [BDG94].

2.5 Verification: Qualitative and Quantitative Methods

Most of the research on the verification of randomized distributed systems is concerned with properties that hold with probability 1. The advantage of such properties is that for finite state processes they do not depend on the actual probabilities of the transitions, but rather on whether those transitions have probability 0 or probability different from 0. Thus, the problem of checking whether a system satisfies a property with probability 1 is reduced to the problem of checking whether a non-randomized system satisfies some other property. This method is called *qualitative*, as opposed to the *quantitative* method, where probabilities different from 1 also matter.

The rationale behind the qualitative method is that a randomized process, rather than always guaranteeing success, usually guarantees success with probability 1, which is practically the same as guaranteeing success always. The quantitative method becomes relevant whenever a system has infinitely many states or the complexity of an algorithm needs to be studied.

Almost all the papers that we describe in this section are based on a model where n Markov chains evolve concurrently. Each Markov chain represents a process, and the pure nondeterminism arises from the choice of what Markov chain performs the next transition (what process is scheduled next). The object that resolves the nondeterminism is called a *scheduler* or *adversary*, and the result of a scheduler on a collection of concurrent Markov chains is a new Markov chain that describes one of the possible evolutions of the global system. Usually a scheduler is required to be *fair* in the sense that each process should be scheduled infinitely many times.

2.5.1 Qualitative Method: Proof Techniques

Huart, Sharir and Pnueli [HSP83] consider n finite state asynchronous randomized processes that run in parallel, and provide two necessary and sufficient conditions to guarantee that a given set of goal states is reached with probability 1 under any fair scheduler. A scheduler is the entity that at any point chooses the next process that performs a transition. The result of the action of a scheduler on n processes is a Markov chain, on which it is possible to study probabilities. A scheduler is fair if and only if, for each path in the corresponding Markov chain, each process is scheduled infinitely many times. The authors show that in their model

each property described by reaching a collection of states has either probability 0 or probability 1. Then, they describe a decision procedure for the *almost sure* reachability of a set of goal states. The procedure either constructs a decomposition of the state space into a sequence of components with the property that any fair execution of the program must move down the sequence with probability 1 until it reaches the goal states (goal states reached with probability 1), or finds an *ergodic* set of states through which the program can loop forever with probability 1 (goal states reached with probability 0). Finally the authors give some examples of problems where the use of randomization does not provide any extra power over pure nondeterminism. The proof principle of [HSP83] is generalized to the infinite state case by Hart and Sharir [HS85].

Lehmann and Shelah [LS82] extend the temporal logic of linear time of Pnueli [Pnu82] to account for properties that hold with probability 1, and they provide three complete axiomatizations of the logic: one axiomatization is for general models, one is for finite models, and one is for models with bounded transition probabilities (same as the minimum probability requirement of Larsen and Skou [LS91]). A model of the logic is essentially a Markov chain, or alternatively an unlabeled generative process. The logic of Lehmann and Shelah [LS82] is obtained from the logic of Pnueli [Pnu82] by adding a new modal operator ∇ whose meaning is that the argument formula is satisfied with probability 1.

Pnueli [Pnu83] introduces the notion of *extreme fairness* and shows that a property that holds for all extreme fair executions holds with probability 1. Furthermore, Pnueli presents a sound proof rule based on extreme fairness and linear temporal logic. The model consists of n randomized processes in parallel. Each process is a state machine where each state enables a probabilistic transition, which lead to several *modes*. Resolving the nondeterminism leads to a Markov chain. However, only those Markov chains that originate from fair scheduling policies are considered. Then, an execution (a path in the Markov chain) is extremely fair relative to a property ϕ (ϕ is a property that is satisfied by states) if and only if for each transition that occurs infinitely many times from states that satisfy ϕ, each mode of the transition occurs infinitely many times. An execution is extremely fair if and only if it is extremely fair relative to any formula ϕ expressed in the logic used in [Pnu83]. The proof rule of Pnueli [Pnu83], along with some other new rules, is used by Pnueli and Zuck [PZ86] to verify two non-trivial randomized algorithms, including the Randomized Dining Philosophers algorithm of Lehmann and Rabin [LR81]. Zuck [Zuc86] introduces the notion of α-fairness and shows that α-fairness is complete for temporal logic properties that hold with probability 1.

Rao [Rao90] extends UNITY [CM88] to account for randomized systems and properties that hold with probability 1. The main emphasis is on properties rather than states. A new notion of *weak probabilistic precondition* is introduced that, together with the extreme fairness of Pnueli, generalizes weakest preconditions. Finally, based on the work of Huart et al. [HSP83], Rao argues that his new logic is complete for finite state programs.

2.5.2 Qualitative Method: Model Checking

Vardi [Var85] presents a method for deciding whether a probabilistic concurrent finite state program satisfies a linear temporal logic specification, where satisfaction means that a formula is satisfied with probability 1 whenever the scheduler is fair. A program is given as a *Concurrent Markov Chain*, which is a transition system with *nondeterministic* and *probabilistic* states. A

subset F of the nondeterministic states is called the set of *fair* states. A scheduler is a function that, based on the past history of a program, chooses the next transition to perform from a nondeterministic state. The result of the action of a scheduler on a program is a Markov chain on which it is possible to study the probability that some linear temporal logic formula is satisfied. A path in the Markov chain is *fair* if for each fair state that occurs infinitely many times each one of the possible nondeterministic choices from that state occurs infinitely many times; a scheduler is *fair* if the fair paths have probability 1 in the corresponding Markov chain. The model checking algorithm of Vardi works in time polynomial in the size of the program and doubly exponential in the size of the specification. By considering a slightly restricted logic, Vardi and Wolper [VW86] reduce the complexity of the model checking algorithm to only one exponent in the size of the formula.

Courcoubetis and Yannakakis [CY88, CY90] investigate the complexity of model checking *linear time propositional temporal logic* of sequential and concurrent probabilistic processes. A sequential process is a Markov chain and a concurrent process is a Concurrent Markov Chain. They give a model checking algorithm that runs in time linear in the size of the program and exponential in the size of the formula, and they show that the problem is in PSPACE. Moreover, they give an algorithm for computing the exact probability with which a sequential program satisfies a formula.

Alur, Courcoubetis and Dill [ACD91a, ACD91b] develop a model checking algorithm for probabilistic real-time systems. Processes are modeled as a *generalized semi-Markov process*, which are studied in [Whi80, She87]. Essentially a process is a finite state transition system with timing constraints expressed by probability distributions on the delays. They impose the restriction that every distribution is either discrete, or exponential, or has a density function which is different from 0 only on a finite collection of intervals (in [ACD91a] only this last case is studied). The temporal logic, called TCTL, is an extension of the branching-time temporal logic of Emerson and Clarke [EC82] where time delays are added to the modal operators. TCTL can detect only whether a formula is satisfied with probability 0, or with a positive probability, or with probability 1. The model checking algorithm transforms a process into a finite state process without probabilities and real-time, thus allowing the use of other existing algorithms. The problem of model-checking for TCTL is PSPACE-hard.

2.5.3 Quantitative Method: Model Checking

Hansson [Han91, Han94] defines a model checking algorithm for his Labeled Concurrent Markov Chain model and his branching-time temporal logic TPCTL. Time is discrete in Hansson's model, but the logic improves on previous work because probabilities can be quantified (i.e., probabilities can be between 0 and 1). The previous model checking algorithms relied heavily on the fact that probabilities were not quantified. The algorithm is based on the algorithm for model checking of Clarke, Emerson and Sistla [CES83], and on previous work of Hansson and Jonsson [HJ89] where a model checking algorithm for PCTL (TPCTL without time) is presented. In order to deal with quantified probabilities, the algorithm reduces the computation of the probability of an event to a collection of finitely many linear recursive equations. The algorithm has an exponential complexity; however, Hansson shows that for a large class of interesting problems the algorithm is polynomial.

Chapter 3

Preliminaries

3.1 Probability Theory

The rigorous study of randomized algorithms requires the use of several probability measures. This section introduces the basic concepts of measure theory that are necessary. Most of the results are taken directly from Halmos [Hal50] and Rudin [Rud66], and the proofs can be found in the same books or in any other good book on measure theory or probability theory.

3.1.1 Measurable Spaces

Consider a set Ω. A *field* on Ω, denoted by F, is a family of subsets of Ω that contains Ω, and that is closed under complementation and finite union. A *σ-field* on Ω, denoted by $\mathcal{F}$, is a field on Ω that is closed under countable union. The elements of a σ-field are called *measurable sets*. The pair $(\Omega, \mathcal{F})$ is called a *measurable space*.

A field *generated* by a family of sets $\mathcal{C}$, denoted by $F(\mathcal{C})$, is the smallest field that contains $\mathcal{C}$. The σ-field generated by a family of sets $\mathcal{C}$, denoted by $\sigma(\mathcal{C})$, is the smallest σ-field that contains $\mathcal{C}$. The family $\mathcal{C}$ is called a *generator* for $\sigma(\mathcal{C})$. A trivial property of a generator $\mathcal{C}$ is $\sigma(\mathcal{C}) = \sigma(F(\mathcal{C}))$.

The field generated by a family of sets can be obtained following a simple procedure.

Proposition 3.1.1 *Let $\mathcal{C}$ be a family of subsets of Ω.*

1. *Let $F_1(\mathcal{C})$ be the family containing $\emptyset$, Ω, and all $C \subseteq \Omega$ such that $C \in \mathcal{C}$ or $(\Omega - C) \in \mathcal{C}$.*
2. *Let $F_2(\mathcal{C})$ be the family containing all finite intersections of elements of $F_1(\mathcal{C})$.*
3. *Let $F_3(\mathcal{C})$ be the family containing all finite unions of disjoint elements of $F_2(\mathcal{C})$.*

Then $F(\mathcal{C}) = F_3(\mathcal{C})$. ■

3.1.2 Probability Measures and Probability Spaces

Let $\mathcal{C}$ be a family of subsets of Ω. A measure μ on $\mathcal{C}$ is a function that assigns a non-negative real value (possibly ∞) to each element of $\mathcal{C}$, such that

1. if $\emptyset$ is an element of $\mathcal{C}$, then $\mu(\emptyset) = 0$.

2. if $(C_i)_{i \in N}$ forms a sequence of pairwise disjoint elements of $\mathcal{C}$, and $\cup_i C_i$ is an element of $\mathcal{C}$, then $\mu(\cup_i C_i) = \sum_i \mu(C_i)$.

The last property is called *σ-additivity*. If $(\Omega, \mathcal{F})$ is a measurable space, then a measure on $\mathcal{F}$ as called a measure on $(\Omega, \mathcal{F})$.

A measure on a family of sets $\mathcal{C}$ is *finite* if the measure of each element of $\mathcal{C}$ is finite.

A *measure space* is a triple $(\Omega, \mathcal{F}, \mu)$, where $(\Omega, \mathcal{F})$ is a measurable space, and μ is a measure on $(\Omega, \mathcal{F})$. A measure space $(\Omega, \mathcal{F}, \mu)$ is *complete* iff for each element C of $\mathcal{F}$ such that $\mu(C) = 0$, each subset of C is measurable and has measure 0, i.e., for each $C' \subset C$, $C' \in \mathcal{F}$ and $\mu(C') = 0$. A measure space is *discrete* if $\mathcal{F}$ is the power set of Ω and the measure of each measurable set is the sum of the measures of its points. Discrete spaces will play a fundamental role in our theory.

A *probability space* is a triple $(\Omega, \mathcal{F}, P)$, where $(\Omega, \mathcal{F})$ is a measurable space, and P is a measure on $(\Omega, \mathcal{F})$ such that $P(\Omega) = 1$. The measure P is also referred to as a *probability measure* or a *probability distribution*. The set Ω is called the *sample space*, and the elements of $\mathcal{F}$ are called *events*. We denote a generic event by E, possibly decorated with primes and indices. A standard convention with probability measures and event is that the measure of an event is denoted by $P[E]$ rather than by $P(E)$.

3.1.3 Extensions of a Measure

The following two theorems shows methods to extend a measure defined on a collection of sets. The first theorem says that it is possible to define a probability measure P on a measurable space $(\Omega, \mathcal{F})$ by specifying P only on a generator of $\mathcal{F}$; the second theorem states that every measure space can be extended to a complete measure space.

Thus, from the first theorem we derive that in order to check the equality of two probability measures P_1 and P_2 on $(\Omega, \mathcal{F})$, it is enough to compare the two measures on a field that generates $\mathcal{F}$.

Theorem 3.1.2 (Extension theorem) *A finite measure μ on a field F has a unique extension to the σ-field generated by F. That is, there exists a unique measure $\bar{\mu}$ on $\sigma(F)$ such that for each element C of F, $\bar{\mu}(C) = \mu(C)$.* ■

Theorem 3.1.3 *Let $(\Omega, \mathcal{F}, \mu)$ be a measure space. Let $\mathcal{F}'$ be the set of subsets of Ω of the form $C \cup N$ such that $C \in \mathcal{F}$ and N is a subset of a set of measure 0 in $\mathcal{F}$. Then, $\mathcal{F}'$ is a σ-field. Furthermore, the function μ' defined by $\mu'(C \cup N) = \mu(C)$ is a complete measure on $\mathcal{F}'$. We denote the measure space $(\Omega, \mathcal{F}', \mu')$ by completion$((\Omega, \mathcal{F}, \mu))$.* ■

3.1.4 Measurable Functions

Let $(\Omega, \mathcal{F})$ and $(\Omega', \mathcal{F}')$ be two measurable spaces. 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 $\mathcal{F}$. The next proposition shows that the measurability of f can be checked just by analyzing a generator of $\mathcal{F}'$.

Proposition 3.1.4 *Let $(\Omega, \mathcal{F})$ and $(\Omega', \mathcal{F}')$ be two measurable spaces, and let $\mathcal{C}$ be a generator of $\mathcal{F}'$. Let f be a function form Ω to Ω'. Then f is measurable iff for each element C of $\mathcal{C}$, the inverse image $f^{-1}(C)$ is an element of $\mathcal{F}$.* ■

Another property that we need is the closure of measurable functions under composition.

Proposition 3.1.5 *Let f be a measurable function from $(\Omega_1, \mathcal{F}_1)$ to $(\Omega_2, \mathcal{F}_2)$, and let g be a measurable function from $(\Omega_2, \mathcal{F}_2)$ to $(\Omega_3, \mathcal{F}_3)$. Then $f \circ g$ is a measurable function from $(\Omega_1, \mathcal{F}_1)$ to $(\Omega_3, \mathcal{F}_3)$.* ■

3.1.5 Induced Measures and Induced Measure Spaces

Proposition 3.1.6 *Let f be a measurable function from $(\Omega, \mathcal{F})$ to $(\Omega', \mathcal{F}')$, and let μ be a measure on $(\Omega, \mathcal{F})$. Let μ' be defined on $\mathcal{F}'$ as follows: for each element C of $\mathcal{F}'$, $\mu'(C) = \mu(f^{-1}(C))$. Then μ' is a measure on $(\Omega', \mathcal{F}')$. The measure μ' is called the measure* induced *by f, and is denoted by $f(\mu)$.* ■

Based on the result above, it is possible to transform a measure space using a function f. Let $(\Omega, \mathcal{F}, \mu)$ be a measure space, and let f be a function defined on Ω. Let Ω' be $f(\Omega)$, and let $\mathcal{F}'$ be the set of subsets C of Ω' such that $f^{-1}(C) \in \mathcal{F}$. Then, $\mathcal{F}'$ is a σ-field, and f is a measurable function from $(\Omega, \mathcal{F})$ to $(\Omega', \mathcal{F}')$. Thus, the space $(\Omega', \mathcal{F}', f(\mu))$ is a measure space. We call such a space the space *induced* by f, and we denote it by $f((\Omega, \mathcal{F}, \mu))$. Observe that if $(\Omega, \mathcal{F}, \mu)$ is a probability space, then $f((\Omega, \mathcal{F}, \mu))$ is a probability space as well, and that induced measure spaces preserve discreteness and completeness.

3.1.6 Product of Measure Spaces

Let $(\Omega_1, \mathcal{F}_1)$ and $(\Omega_2, \mathcal{F}_2)$ be two measurable spaces. Denote by $\mathcal{F}_1 \otimes \mathcal{F}_2$ the σ-field generated by the set of *rectangles* $\{C_1 \times C_2 \mid C_1 \in \mathcal{F}_1, C_2 \in \mathcal{F}_2\}$. The *product space* of $(\Omega_1, \mathcal{F}_1)$ and $(\Omega_2, \mathcal{F}_2)$, denoted by $(\Omega_1, \mathcal{F}_1) \otimes (\Omega_2, \mathcal{F}_2)$, is the measurable space $(\Omega_1 \times \Omega_2, \mathcal{F}_1 \otimes \mathcal{F}_2)$.

Proposition 3.1.7 *Let $(\Omega_1, \mathcal{F}_1, \mu_1)$ and $(\Omega_2, \mathcal{F}_2, \mu_2)$ be two measure spaces where μ_1 and μ_2 are finite measures. Then there is a unique measure, denoted by $\mu_1 \otimes \mu_2$, on $\mathcal{F}_1 \otimes \mathcal{F}_2$ such that for each $C_1 \in \mathcal{F}_1$ and $C_2 \in \mathcal{F}_2$, $\mu_1 \otimes \mu_2(C_1 \times C_2) = \mu_1(C_1)\mu_2(C_2)$.* ■

The *product measure space* of two measure spaces $(\Omega_1, \mathcal{F}_1, \mu_1)$ and $(\Omega_2, \mathcal{F}_2, \mu_2)$, denoted by $(\Omega_1, \mathcal{F}_1, \mu_1) \otimes (\Omega_2, \mathcal{F}_2, \mu_2)$, is the measure space $(\Omega_1 \times \Omega_2, \mathcal{F}_1 \otimes \mathcal{F}_2, \mu_1 \otimes \mu_2)$. It is easy to check that if $(\Omega_1, \mathcal{F}_1, \mu_1)$ and $(\Omega_2, \mathcal{F}_2, \mu_2)$ are probability spaces, then their product is a probability space as well.

The product of two measure spaces is invertible. Let $(\Omega, \mathcal{F}, \mu) = (\Omega_1, \mathcal{F}_1, \mu_1) \otimes (\Omega_2, \mathcal{F}_2, \mu_2)$, and let π_i, $i = 1, 2$, be a *projection* function from $\Omega_1 \times \Omega_2$ to Ω_i, that maps each pair (x_1, x_2) to x_i. Let $\Omega_i' = \pi_i(\Omega_i)$, and let $\mathcal{F}_i' = \{C \mid \pi_i^{-1}(C) \in \mathcal{F}_i\}$. Then $(\Omega_i', \mathcal{F}_i') = (\Omega_i, \mathcal{F}_i)$, and π_i is a measurable function from $(\Omega, \mathcal{F})$ to $(\Omega_i', \mathcal{F}_i')$. The measure $\pi_i(\mu)$ coincides with μ_i, since for each $C \in \mathcal{F}_1$, $\pi_1^{-1}(C) = C \times \Omega_2$, and for each $C \in \mathcal{F}_2$, $\pi_2^{-1}(C) = \Omega_1 \times C$. Thus, the projection of $(\Omega, \mathcal{F}, \mu)$ onto its i^{th} component is $(\Omega_i, \mathcal{F}_i, \mu_i)$.

3.1.7 Combination of Discrete Probability Spaces

In our theory there are several situations in which a discrete probability space is chosen according to some probability distribution, and then an element from the chosen probability space

is chosen according to the corresponding probability distribution. The whole process can be described by a unique probability space.

Let $\{(\Omega_i, \mathcal{F}_i, P_i)\}_{i\geq 0}$ be a family of discrete probability spaces, and let $\{p_i\}_{i\geq 0}$ be a family of real numbers between 0 and 1 such that $\sum_{i\geq 0} p_i = 1$. Define $\sum_{i\geq 0}(\Omega_i, \mathcal{F}_i, P_i)$ to be the triple $(\Omega, \mathcal{F}, P)$, where $\Omega = \cup_{i\geq 0}\Omega_i$, $\mathcal{F} = 2^{\Omega}$, and, for each $x \in \Omega$, $P[x] = \sum_{i\geq 0 | x\in\Omega_i} p_i P_i[x]$. It is easy to verify that $(\Omega, \mathcal{F}, P)$ is a probability space.

The process described by $(\Omega, \mathcal{F}, P)$ is the following: a probability space $(\Omega_i, \mathcal{F}_i, P_i)$ is drawn from $\{(\Omega_i, \mathcal{F}_i, P_i)\}_{i\geq 0}$ with probability p_i, and then an element x is drawn drom Ω_i with probability $P_i[x]$.

3.1.8 Conditional Probability

Let $(\Omega, \mathcal{F}, P)$ be a probability space, and let E be an element of $\mathcal{F}$. Frequently, we need to study the probability of an event E' of $\mathcal{F}$ knowing that event E has occurred. For example, we may want to study the probability that a dice rolled 6 knowing that it rolled a number greater than 3. The probability of a *conditional* event is expressed by $P[E'|E]$. If $P[E] = 0$, then $P[E'|E]$ is undefined; if $P[E] > 0$, then $P[E'|E]$ is defined to be $P[E \cap E']/P[E]$.

Suppose that $P[E] > 0$, and consider the triple $(\Omega|E, \mathcal{F}|E, P|E)$ where $\Omega|E = E$, $\mathcal{F}|E = \{E' \cap E \mid E' \in \mathcal{F}\}$, and for each event E' of $\mathcal{F}|E$, $P|E[E'] = P[E'|E]$. Then it is easy to show that $(\Omega|E, \mathcal{F}|E, P|E)$ is a probability space. We call this space a *conditional probability space*.

Conditional measures give us an alternative way to express the probability of the intersection of several events. That is,

$$P[E_1 \cap \cdots \cap E_n] = P[E_1]P[E_2|E_1]\cdots P[E_n|E_1 \cap \cdots \cap E_{n-1}].$$

If $P[E'] = P[E'|E]$, then $P[E \cap E'] = P[E]P[E']$. In this case the events E and E' are said to be *independent*.

3.1.9 Expected Values

Let $(\Omega, \mathcal{F})$ be a measurable space, and let $(\Re, \mathcal{R})$ be the measurable space where $\Re$ is the set of real numbers, and $\mathcal{R}$ is the σ-field generated by the open sets of the real line. A *random variable* on $(\Omega, \mathcal{F})$, denoted by X, is a measurable function from $(\Omega, \mathcal{F})$ to $(\Re, \mathcal{R})$.

We use random variables to deal with timed systems. An example of a random variable is the function that, given a computation of a system, returns the time it takes to the system to achieve a goal in the given computation. In our case, the computations of a system are chosen at random, and thus, a natural estimate of the performance of the system is the average time it takes to the system to achieve the given goal.

The above idea is expressed formally by the *expected value* of a random variable, which is a weighted average of X. Specifically, let $(\Omega, \mathcal{F}, P)$ be a probability space, and let X be a random variable on $(\Omega, \mathcal{F})$. Then the *expected value* of X, denoted by $E[X]$, is the weighted average of X based on the probability distribution P. We do not show how to compute the expected value of a random variable in general, and we refer the interested reader to [Hal50]. Here we just mention that if Ω can be partitioned in a countable collection of measurable sets $(C_i)_{i\geq 0}$ such that for each set C_i, $X(C_i)$ is a singleton, then $E[X] = \sum_{i\geq 0} P[C_i]X(c_i)$, where for each i c_i is an element of $\mathcal{F}_i$.

3.1.10 Notation

Throughout the thesis we adopt some conventional notation concerning probability spaces. We use the notation $\mathcal{P}$, possibly decorated with indexes and primes, to denote a generic probability space. Thus, the expression $\mathcal{P}_i'$ stands for the probability space $(\Omega_i', \mathcal{F}_i', P_i')$. Furthermore, if a generic expression *exp* denotes a probability space $(\Omega, \mathcal{F}, P)$, we use Ω_{exp}, $\mathcal{F}_{exp}$, and P_{exp} to denote Ω, $\mathcal{F}$, and P, respectively.

If $(\Omega, \mathcal{F}, P)$ is a probability space, and E is a generic set, we use $P[E]$ to denote $P[E \cap \Omega]$. If $E \cap \Omega$ is not an element of $\mathcal{F}$, then $P[E]$ is undefined.

A special kind of probability space is a probability space with a unique element in its sample set. The corresponding measure is called a *Dirac* distribution. We use the notation $\mathcal{D}(x)$ to denote a probability space $(\Omega, \mathcal{F}, P)$ where $\Omega = \{x\}$.

Another important kind of probability space is a space with finitely many elements, each one with the same probability. The corresponding measure is called a *uniform* distribution. We use the notation $\mathcal{U}(x_1, \ldots, x_n)$ to denote a discrete probability space $(\Omega, \mathcal{F}, P)$ where $\Omega = \{x_1, \ldots, x_n\}$ and, for each element x_i of Ω, $P[x_i] = 1/n$.

In the thesis we use heavily discrete probability spaces with no 0-probability elements. It is easy to verify that the sample set of these probability spaces is at most countable. If C is any set, then we denote by $Probs(C)$ the set of discrete probability spaces $(\Omega, \mathcal{F}, P)$ with no 0-probability elements such that $\Omega \subseteq C$.

3.2 Labeled Transition Systems

A Labeled Transition System [Kel76, Plo81] is a state machine with labeled transitions. The labels, also called *actions*, are used to model communication between a system and its external environment. Labeled transition systems have been used successfully for the analysis of concurrent and distributed systems [DH84, Mil89, LT87, LV93a]; for this reason we choose them as our basic model.

Currently there are several definitions of labeled transition systems, each one best suited for the kind of application it is meant for. In this section we present a definition of labeled transition systems in the style of [LV93a].

3.2.1 Automata

An *automaton* A consists of four components:

1. a set $states(A)$ of states.
2. a nonempty set $start(A) \subseteq states(A)$ of start states.
3. an action signature $sig(A) = (ext(A), int(A))$, where $ext(A)$ and $int(A)$ are disjoint sets of *external* and *internal* actions, respectively. Denote by $acts(A)$ the set $ext(A) \cup int(A)$ of actions.
4. a transition relation $trans(A) \subseteq states(A) \times acts(A) \times states(A)$. The elements of $trans(A)$ are referred to as *transitions* or *steps*.

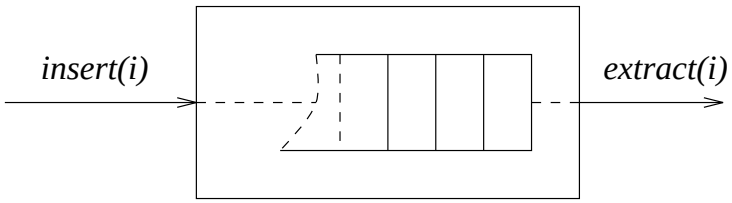


Figure 3-1: The *Buffer* automaton.

Thus, an automaton is a labeled transition system, possibly with multiple start states, whose actions are partitioned into external and internal actions. The external actions model communication with the external environment; the internal actions model internal communication, not visible from the external environment.

We use s to denote a generic state, and a and b to denote a generic action. We also use τ to denote a generic internal action. All our conventional symbols may be decorated with primes and indexes. We say that an action a is enabled from a state s in A if there exists a state s' of A such that (s, a, s') is a transition of A.

A standard alternative notation for transitions is $s \xrightarrow{a} s'$. This notation can be extended to finite sequences of actions as follows: $s \xrightarrow{a_1 \cdots a_n} s'$ iff there exists a sequence of states $s_1, \ldots, s_{n-1}$ such that $s \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots s_{n-1} \xrightarrow{a_n} s_n$. To abstract from internal computation, there is another standard notion of *weak transition*, denoted by $s \stackrel{a}{\Longrightarrow} s'$. The action a must be external, and the meaning of $s \stackrel{a}{\Longrightarrow} s'$ is that there are two finite sequences β_1, β_2 of internal actions such that $s \xrightarrow{\beta_1 a \beta_2} s'$. As for ordinary transitions, weak transitions can be generalized to finite sequences of external actions. A special case is given by the empty sequence: $s \Longrightarrow s'$ iff either $s' = s$ or there exists a finite sequence β of internal actions such that $s \xrightarrow{\beta} s'$.

Example 3.2.1 A classic example of an automaton is an unbounded ordered buffer that stores natural numbers (see Figure 3-1). An external user sends natural numbers to the buffer, and the buffer sends back to the external environment the ordered sequence of numbers it receives from the user.

The automaton *Buffer* of Figure 3-1 can be described as follows. All the actions of *Buffer* are external and are of the form $insert(i)$ and $extract(i)$, where i is a natural number, i.e., the actions of *Buffer* are given by the infinite set $\cup_{i \in N}\{insert(i), extract(i)\}$. The states of *Buffer* are the finite sequences of natural numbers, and the start state of *Buffer* is the empty sequence. The actions of the form $insert(i)$ are enabled from every state of *Buffer*, i.e., for each state s and each natural number i there is a transition $(s, insert(i), is)$ in *Buffer*, where is denotes the sequence obtained by appending i to the left of s. The actions of the form $extract(i)$ are enabled only from those states where i is the rightmost element in the corresponding sequence of numbers, i.e., for each state s and each natural number i there is a transition $(si, extract(i), s)$ of *Buffer*. No other transitions are defined for *Buffer*.

Observe that from every state of *Buffer* there are infinitely many actions enabled. The way to choose among those actions is not specified in *Buffer*. In other words, the choice of the transition to perform is nondeterministic. In this case the nondeterminism models the arbitrary behavior of the environment.

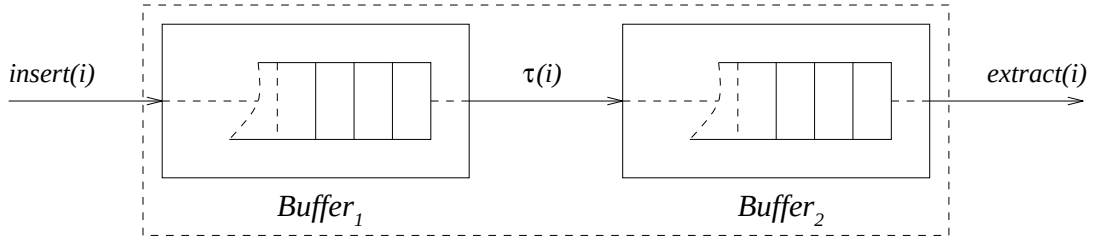


Figure 3-2: Concatenation of two buffers.

The role of internal actions becomes clear when we concatenate two buffers as in Figure 3-2. The communication that occurs between the two buffers is internal in the sense that it does not affect directly the external environment. Another useful observation about the concatenation of the two buffers in Figure 3-2 is that nondeterminism expresses two different phenomena: the arbitrary behavior of the environment, and the arbitrary scheduling policy that can be adopted in choosing whether $Buffer_1$ or $Buffer_2$ performs the next transition. In general nondeterminism can express even a third phenomenon, namely, the fact that an arbitrary state can be reached after the occurrence of an action. Such a form of nondeterminism would arise if we assume that a buffer may lose data by failing to modify its state during an insertion operation. ■

3.2.2 Executions

The evolution of an automaton can be described by means of its executions. An *execution fragment* α of an automaton A is a (finite or infinite) sequence of alternating states and actions starting with a state and, if the execution fragment is finite, ending in a state

$$\alpha = s_0 a_1 s_1 a_2 s_2 \cdots$$

where for each i, (s_i, a_{i+1}, s_{i+1}) is a transition of A. Thus, an execution fragment represents a possible way to resolve the nondeterminism in an automaton.

Denote by $fstate(\alpha)$ the first state of α and, if α is finite, denote by $lstate(\alpha)$ the last state of α. Furthermore, denote by $frag^*(A)$ and $frag(A)$ the sets of finite and all execution fragments of A, respectively.

An *execution* is an execution fragment whose first state is a start state. Denote by $exec^*(A)$ and $exec(A)$ the sets of finite and all execution of A, respectively. A state s of A is *reachable* if there exists a finite execution of A that ends in s.

The length of an execution fragment α, denoted by $|\alpha|$, is the number of actions that occur in α. If α is infinite, then $|\alpha| = \infty$.

A finite execution fragment $\alpha_1 = s_0 a_1 s_1 \cdots a_n s_n$ of A and an execution fragment $\alpha_2 = s_n a_{n+1} s_{n+1} \cdots$ of A can be *concatenated*. In this case the concatenation, written $\alpha_1 \frown \alpha_2$, is the execution fragment $s_0 a_1 s_1 \cdots a_n s_n a_{n+1} s_{n+1} \cdots$. If $\alpha = \alpha_1 \frown \alpha_2$, then we denote α_2 by $\alpha \triangleright \alpha_1$ (read "α after α_1").

An execution fragment α_1 of A is a *prefix* of an execution fragment α_2 of A, written $\alpha_1 \leq \alpha_2$, if either $\alpha_1 = \alpha_2$ or α_1 is finite and there exists an execution fragment α_1' of A such that $\alpha_2 = \alpha_1 \frown \alpha_1'$. The execution fragment α_1' is also called a *suffix* of α_2 and is denoted by $\alpha_2 \triangleright \alpha_1$.

3.2.3 Traces

The executions of an automaton contain a lot of information that is irrelevant to the environment, since the interaction between an automaton and its environment occurs through external actions only. The trace of an execution is the object that represents the actual interaction that occurs between an automaton and its environment during an execution.

The *trace* of an execution (fragment) α of an automaton A, written $trace_A(\alpha)$, or just $trace(\alpha)$ when A is clear, is the list obtained by restricting α to the set of external actions of A, i.e., $trace(\alpha) = \alpha \restriction ext(A)$. We say that β is a trace of an automaton A if there exists an execution α of A with $trace(\alpha) = \beta$. Denote by $traces^*(A)$ and $traces(A)$ the sets of finite and all traces of A, respectively. Note, that a finite trace can be the trace of an infinite execution.

3.2.4 Trace Semantics

In [LV93a] automata are compared based on traces. Specifically, a preorder relation is defined between automata based on inclusion of their traces:

$$A_1 \sqsubseteq_T A_2 \text{ iff } traces(A_1) \subseteq traces(A_2).$$

The *trace preorder* can express a notion of implementation, usually referred to as a *safe* implementation. That is, A_1, the implementation, cannot do anything that is forbidden by A_2, the specification. For example, no implementation of the buffer of Figure 3-1 can return natural numbers that were never entered or natural numbers in the wrong order.

Although the trace preorder is weak as a notion of implementation, and so finer relations could be more appropriate [DeN87, Gla90, Gla93], there are several situations where a trace based semantics is sufficient [LT87, Dil88, AL93, GSSL94]. The advantage of a trace based semantics is that it is easy to handle.

In this thesis we concentrate mainly on trace based semantics; however, the techniques that we develop can be extended to other semantic notions as well.

3.2.5 Parallel Composition

Parallel composition is the operator on automata that identifies how automata communicate and synchronize. There are two main synchronization mechanisms for labeled transition systems, better known as the CCS synchronization style [Mil89], and the CSP synchronization style [Hoa85]. In the CCS synchronization style the external actions are grouped in pairs of *complementary* actions; a synchronization occurs between two automata that perform complementary actions, and becomes invisible to the external environment, i.e., a synchronization is an internal action. Unless specifically stated through an additional *restriction* operator, an automaton is allowed not to synchronize with another automaton even though a synchronization is possible. In the CSP synchronization style two automata must synchronize on their common actions and evolve independently on the others. Both in the CCS and CSP styles, communication is achieved through synchronization.

In this thesis we adopt the CSP synchronization style, which is essentially the style adopted in [LT87, Dil88, LV93a]. A technical problem that arises in our framework is that automata may communicate through their internal actions, while internal actions are not supposed to be visible. To avoid these unwanted communications, we define a notion of compatibility between

automata. Two automata A_1, A_2 are *compatible* iff $int(A_1) \cap acts(A_2) = \emptyset$ and $acts(A_1) \cap int(A_2) = \emptyset$.

The *parallel composition* of two compatible automata A_1, A_2, denoted by $A_1 \| A_2$, is the automaton A such that

1. $states(A) = states(A_1) \times states(A_2)$.
2. $start(A) = start(A_1) \times start(A_2)$.
3. $sig(A) = (ext(A_1) \cup ext(A_2), int(A_1) \cup int(A_2))$.
4. $((s_1, s_2), a, (s_1', s_2')) \in trans(A)$ iff
   (a) if $a \in acts(A_1)$, then $(s_1, a, s_1') \in trans(A_1)$, else $s_1' = s_1$, and
   (b) if $a \in acts(A_2)$, then $(s_2, a, s_2') \in trans(A_2)$, else $s_2' = s_2$.

If two automata are incompatible and we want to compose them in parallel, the problem can be solved easily by renaming the internal actions of one of the automata. The renaming operation is simple: just rename each occurrence of each action in the action signature and the transition relation of the given argument automaton. At this point it is possible to understand how to build a system like the one described in Figure 3-2. $Buffer_1$ is obtained from *Buffer* by renaming the actions $extract(i)$ into $\tau(i)$, and $Buffer_2$ is obtained from *Buffer* by renaming the actions $insert(i)$ into $\tau(i)$. Then, $Buffer_1$ and $Buffer_2$ are composed in parallel, and finally the actions $\tau(i)$ are made internal. This last step is achieved through a *Hide* operation, whose only effect is to change the signature of an automaton.

We conclude by presenting two important properties of parallel composition. The first property concerns projections of executions. Let $A = A_1 \| A_2$, and let (s_1, s_2) be a state of A. Let i be either 1 or 2. The *projection* of (s_1, s_2) onto A_i, denoted by $(s_1, s_2) \lceil A_i$, is s_i. Let $\alpha = s_0 a_1 s_1 \cdots$ be an execution of A. The projection of α onto A_i, denoted by $\alpha \lceil A_i$ is the sequence obtained from α by projecting all the states onto A_i, and by removing all the actions not in $acts(A_i)$ together with their subsequent states.

Proposition 3.2.1 *Let $A = A_1 \| A_2$, and let α be an execution of A. Then $\alpha \lceil A_1$ is an execution of A_1 and $\alpha \lceil A_2$ is an execution of A_2.* ■

The projection of an execution of A onto one of the components A_i is essentially the view of A_i of the execution α. In other words the projection represents what A_i does in order for A to produce α. Proposition 3.2.1 states that the view of A_i is indeed something that A_i can do.

The second property concerns the trace preorder.

Proposition 3.2.2 *Let $A_1 \sqsubseteq_T A_1'$. Then, for each A_2 compatible with both A_1 and A_1', $A_1 \| A_2 \sqsubseteq_T A_1' \| A_2$.* ■

The property expressed in Proposition 3.2.2 is better known as *substitutivity* or *compositionality*. In other words $\sqsubseteq_T$ is a precongruence with respect to parallel composition. Substitutivity is one of the most important properties that an implementation relation should satisfy. Informally, substitutivity says that an implementation A_1 of a system A_1' works correctly in any context where A_1' works correctly. Substitutivity is also the key idea at the base of modular verification techniques.

Chapter 4

Probabilistic Automata

4.1 What we Need to Model

Our main goal is to analyze objects that at any point can evolve according to a probability distribution. The simplest example of a random computation is the process of flipping a coin. Thus, a program may contain an instruction like

$$x := \mathit{flip}$$

whose meaning is to assign to x the result of a coin flip. From the state-machine point of view, the transition relation of the corresponding automaton should be specified by giving the states reachable after the coin flip, together with their probability. Thus, the coin flipping process can be represented by the labeled transition system of Figure 4-1. The edges joining two states are associated with an action and a weight, where the weight of an edge is the probability of choosing that specific edge. Thus, we require that for each state that has some outgoing edges, the sum of the weights of the outgoing edges is 1.

However, we also need to deal with nondeterminism. Consider a more complicated process where a coin is flipped, but where the coin can be either fair, i.e., it yields *head* with probability 1/2, or unfair by yielding *head* with probability 2/3. Furthermore, suppose that the process emits a beep if the result of the coin flip is *head*. In this case, the choice of which coin to flip is nondeterministic, while the outcome of the coin flip is probabilistic. The start state should enable two separate transitions, each one corresponding to the flip of a specific coin. Figure 4-2 represents the nondeterministic coin flipping process. The start state enables two separate groups of weighted edges; each group is identified by an arc joining all of its edges, and the edges of each group form a probability distribution.

At this point we may be tempted to ask the following question:

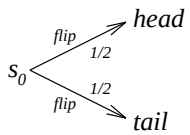


Figure 4-1: The coin flipping process.

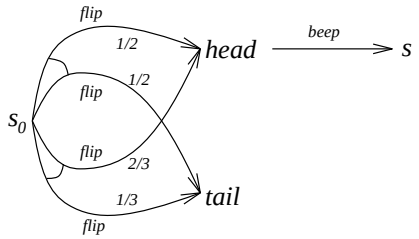


Figure 4-2: The nondeterministic coin flipping process.

"*What is the probability that the nondeterministic coin flipper beeps?*"

The correct answer is

"*It depends on which coin is flipped.*"

Although this observation may appear to be silly, the lesson that we learn is that it is not possible to talk about the probability of some event until the nondeterminism is resolved. Perhaps we could give a more accurate answer as follows:

> "*The probability that the nondeterministic coin flipper beeps is either* $1/2$ *or* $2/3$*, depending on which coin is flipped.*"

However, there are two possible objections. The first objection concerns the way a coin is chosen. What happens if the coin to be flipped is chosen at random? After all, in the definition of the nondeterministic coin flipper there are no limitations to the way a coin is chosen. In this case, the correct answer would be

> "*The probability that the nondeterministic coin flipper beeps is between* $1/2$ *and* $2/3$*, depending on how the coin to be flipped is chosen.*"

The second objection concerns the possibility of scheduling a transition. What happens if the scheduler does not schedule the *beep* transition even though it is enabled? In this case the correct answer would be

> "*Under the hypothesis that some transition is scheduled whenever some transition is enabled, the probability that the nondeterministic coin flipper beeps is between* $1/2$ *and* $2/3$*, depending on how the coin to be flipped is chosen.*"

There is also another statement that can be formulated in relation to the question:

> "*The nondeterministic coin flipper does not beep with any probability greater than* $2/3$."

This last property is better known as a *safety property* [AS85] for ordinary labeled transition systems.

Let us go back to the scheduling problem. There are actual cases where it is natural to allow a scheduler not to schedule any transition even though some transition is enabled. Consider a new nondeterministic coin flipper with two buttons, marked *fair* and *unfair*, respectively. The

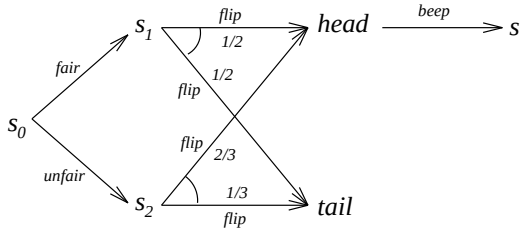


Figure 4-3: The triggered coin flipping process.

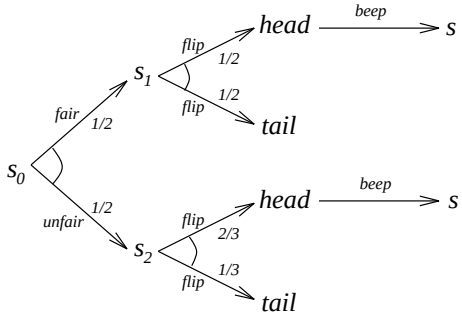


Figure 4-4: A computation of the triggered coin flipping process.

buttons can be pressed by an external user. Suppose that pressing one button disables the other button, and suppose that the fair coin is flipped if the button marked *fair* is pressed, and that the unfair coin is flipped if the button marked *unfair* is pressed. The new process is represented in Figure 4-3. In this case the scheduler models the external environment, and a user may decide not to press any button, thus not scheduling any transition from s_0 even though some transition is enabled. An external user may even decide to flip a coin and press a button only if the coin gives *head*, or flip a coin and press *fair* if the coin gives *head* and press *unfair* if the coin gives *tail*. That is, an external user acts like a scheduler that can use randomization for its choices. If we ask again the question about the probability of beeping, a correct answer would be

> "*Assuming that beep is scheduled whenever it is enabled, the probability that the triggered coin flipper beeps, conditional to the occurrence of a coin flip, is between* 1/2 *and* 2/3."

Suppose now that we resolve all the nondeterminism in the triggered coin flipper of Figure 4-3, and consider the case where the external user presses *fair* with probability 1/2 and *unfair* with probability 1/2. In this case it is possible to study the exact probability that the process beeps, which is 7/12. Figure 4-4 gives a representation of the outcome of the user we have just described. Note that the result of resolving the nondeterminism is not a linear structure as is the case for standard automata, but rather a tree-like structure. This structure is our notion of a *probabilistic execution* and is studied in more detail in Section 4.2.

4.2 The Basic Model

In this section we introduce the basic probabilistic model that is used in the thesis. We formalize the informal ideas presented in Section 4.1, and we extend the parallel composition operator of ordinary automata to the new framework. We also introduce several notational conventions that are used throughout the thesis.

4.2.1 Probabilistic Automata

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. Denote by $acts(M)$ the set $ext(M) \cup int(M)$ of actions.
4. A transition relation $trans(M) \subseteq states(M) \times Probs((acts(M) \times states(M)) \cup \{\delta\})$. Recall from Section 3.1.10 that for each set C, $Probs(C)$ denotes the set of discrete probability spaces $(\Omega, \mathcal{F}, P)$ with no 0-probability elements such that $\Omega \subseteq C$. The elements of $trans(M)$ are referred to as *transitions* or *steps*.

A probabilistic automaton differs from an ordinary automaton only in the transition relation. Each transition represents what in the figures of Section 4.1 is represented by a group of edges joined by an arc. From each state s, once a transition is chosen nondeterministically, the action that is performed and the state that is reached are determined by a discrete probability distribution. Each transition $(s, \mathcal{P})$ may contain a special symbol δ, which represents the possibility for the system not to complete the transition, i.e., to remain in s without being able to engage in any other transition.

Example 4.2.1 (Meaning of δ) To give an idea of the meaning of δ, suppose that M models a person sitting on a chair that stands up with probability 1/2. That is, from the start state s_0 there is a transition of M where one outcome describes the fact that the person stands up and the other outcome describes the fact that the person does not stand up (this is δ). The point is that there is no instant in time where the person decides not to stand up: there are only instants where the person stands up. What the transition leaving s_0 represents is that overall the probability that the person does the action of standing up is 1/2. The need for δ is clarified further in Section 4.2.3, where we study probabilistic executions, and in Section 4.3, where we study parallel composition. ■

The requirement that the probability space associated with a transition be discrete is imposed to simplify the measure theoretical analysis of probabilistic automata. In this thesis we work with discrete probability spaces only, and we defer to further work the extension of the theory to more general probability spaces. The requirement that each transition does not lead to any place with probability 0 is imposed to simplify the analysis of probabilistic automata. All the results of this thesis would be valid even without such a restriction, although the proofs would

contain a lot of uninteresting details. The requirement becomes necessary for the study of live probabilistic automata, which we do not study here.

There are two classes of probabilistic automata that are especially important for our analysis: *simple* probabilistic automata, and *fully probabilistic* automata.

A probabilistic automaton M is *simple* if for each transition $(s, \mathcal{P})$ of $trans(M)$ there is an action a of M 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 it is called a *simple transition with action* a. The probabilistic automata of Figures 4-2 and 4-3 are simple. In a simple probabilistic automaton each transition is associated with a single action and it always completes. The idea is that once a transition is chosen, then only the next state is chosen probabilistically. In this thesis we deal mainly with simple probabilistic automata for a reason that is made clear in Section 4.3. We use general probabilistic automata to analyze the computations of simple probabilistic automata.

A probabilistic automaton M is *fully probabilistic* if M has a unique start state, and from each state of M there is at most one transition enabled. Thus, a fully probabilistic automaton does not contain any nondeterminism. Fully probabilistic automata play a crucial role in the definition of probabilistic executions.

Example 4.2.2 (Probabilistic automata) A probabilistic Turing Machine is a Turing machine with an additional *random tape*. The content of the random tape is instantiated by assigning each cell the result of an independent fair coin flip (say 0 if the coin gives head and 1 if the coin gives tail). If we assume that each cell of the random tape is instantiated only when it is reached by the head of the machine, then a probabilistic Turing machine can be represented as a simple probabilistic automaton. The probabilistic automaton, denoted by M, has a unique internal action τ, and its states are the instantaneous descriptions of the given probabilistic Turing machine; each time the Turing machine moves the head of its random tape on a cell for the first time, M has a probabilistic transition that represents the result of reaching a cell whose content is 0 with probability 1/2 and 1 with probability 1/2.

An algorithm that at some point can flip a coin or roll a dice can be represented as a simple probabilistic automaton where the flipping and rolling operations are simple transitions. If the outcome of a coin flip or dice roll affects the external behavior of the automaton, then the flip and roll actions can be followed by simple transitions whose actions represent the outcome of the random choice. Another possibility is to represent the outcome of the random choice directly in the transition where the random choice is made by performing different actions. In this case the resulting probabilistic automaton would not be simple. Later in the chapter we show why we prefer to represent systems as simple probabilistic automata when possible. ■

4.2.2 Combined Transitions

In Section 4.1 we argued that a scheduler may resolve the nondeterminism using randomization, i.e., a scheduler can generate a new transition by combining several transitions of a probabilistic automaton M. We call the result of the combination of several transitions a *combined transition*. Formally, let M be a probabilistic automaton, and let s be a state of M. Consider a finite or countable set $\{(s, \mathcal{P}_i)\}_{i \in I}$ of transitions of M leaving from s, and a family of non-negative

weights $\{p_i\}_{i \in I}$ such that $\sum_i p_i \leq 1$. Let

$$\mathcal{P} \triangleq \left(\sum_{i \in I | p_i > 0} p_i \mathcal{P}_i \right) + \left(1 - \sum_{i \in I} p_i \right) \mathcal{D}(\delta), \tag{4.1}$$

i.e., $\mathcal{P}$ is a combination of discrete probability spaces as described in Section 3.1.7. The pair $(s, \mathcal{P})$ is called a *combined transition* of M and is denoted by $\sum_{i \in I} p_i(s, \mathcal{P}_i)$. Denote by $ctrans(M)$ the set of combined transitions of M. Note that $trans(M) \subseteq ctrans(M)$.

Thus, the combination of transitions can be viewed as a weighted sum of transitions where the sum of the weights is at most 1. If the sum of the weights is not 1, then nothing is scheduled by default. The reason for δ by default will become clear when we analyze parallel composition in Section 4.3. Note that all the transitions $(s, \mathcal{P}_i)$ where $p_i = 0$ are discarded in Expression (4.1), since otherwise $\mathcal{P}$ would contain elements whose probability is 0. We do not impose the restriction that each p_i is not 0 for notational convenience: in several parts of the thesis the p_i's are given by complex expression that sometimes may evaluate to 0.

Proposition 4.2.1 *The combination of combined transitions of a probabilistic automaton M is a combined transition of M.*

Proof. Follows trivially from the definition of a combined transition. ■

4.2.3 Probabilistic Executions

If we resolve both the nondeterministic and probabilistic choices of a probabilistic automaton, then we obtain an ordinary execution like those usually defined for ordinary automata. Thus, an *execution fragment* of a probabilistic automaton M is a (finite or infinite) sequence of alternating states and actions starting with a state and, if the execution fragment is finite, ending in a state,

$$\alpha = s_0 a_1 s_1 a_2 s_2 \cdots,$$

where for each i there is a transition $(s_i, \mathcal{P}_{i+1})$ of M such that $(a_{i+1}, s_{i+1}) \in \Omega_{i+1}$. Executions, concatenations of executions, and prefixes can be defined as for ordinary automata.

In order to study the probabilistic behavior of a probabilistic automaton, we need a mechanism to resolve only the nondeterminism, and leave the rest unchanged. That is, we need a structure that describes the result of choosing a transition, possibly using randomization, at any point in history, i.e., at any point during a computation. In Figure 4-4 we have given an example of such a structure, and we have claimed that it should look like a tree. Here we give a more significant example to justify such a claim.

Example 4.2.3 (History in a probabilistic execution) Consider a new triggered coin flipper, described in Figure 4-5, that can decide nondeterministically to *beep* or *boo* if the coin flip yields *head*, and consider a computation, described in Figure 4-6, that beeps if the user chooses to flip the fair coin, and boos if the user chooses to flip the unfair coin. Then, it is evident that we cannot identify the two states *head* of Figure 4-6 without reintroducing nondeterminism. In other words, the transition that is scheduled at each point depends on the past history of the system, which is represented by the position of a state in the tree. For a formal definition of a structure like the one of Figure 4-6, however, we need to refer explicitly to the past history of a system. ■

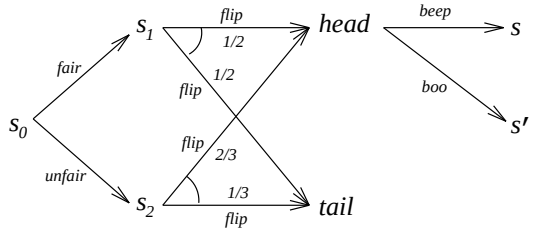


Figure 4-5: The triggered coin flipper with a *boo* sound.

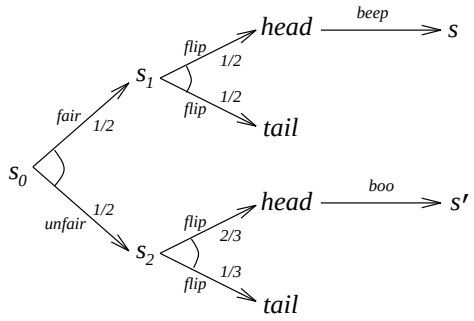


Figure 4-6: A computation of the triggered coin flipper with a *boo* sound.

Let α be a finite execution fragment of a probabilistic automaton M. Define a function $\alpha^\frown$ that applied to a pair (a, s) returns the pair $(a, \alpha a s)$, and applied to δ returns δ. Recall from Section 3.1.5 that the function $\alpha^\frown$ can be extended to probability spaces. Informally, if $(s, \mathcal{P})$ is a combined transition of M and α is a finite execution fragment of M such that $lstate(\alpha) = s$, then the pair $(\alpha, \alpha \frown \mathcal{P})$ denotes a transition of a structure that in its states remembers part of the past history. A *probabilistic execution fragment* of a probabilistic automaton M, is a fully probabilistic automaton, denoted by H, such that

1. $states(H) \subseteq frag^*(M)$. Let q range over states of probabilistic execution fragments.
2. for each transition $(q, \mathcal{P})$ of H there is a combined transition $(lstate(q), \mathcal{P}')$ of M, called the *corresponding combined transition*, such that $\mathcal{P} = q \frown \mathcal{P}'$.
3. each state q of H is reachable in H and enables one transition, possibly $(q, \mathcal{D}(\delta))$.

A *probabilistic execution* is a probabilistic execution fragment whose start state is a start state of M. Denote by $prfrag(M)$ the set of probabilistic execution fragments of M, and by $prexec(M)$ the set of probabilistic executions of M. Also, denote by q_0^H the start state of a generic probabilistic execution fragment H.

Thus, by definition, a probabilistic execution fragment is a probabilistic automaton itself. Condition 3 is technical: reachability is imposed to avoid useless states in a probabilistic execution fragment; the fact that each state enables one transition is imposed to treat uniformly all the points where it is possible not to schedule anything. Figures 4-6 and 4-7 represent two probabilistic executions of the triggered coin flipper of Figure 4-5. The occurrence of δ is represented by a dashed line labeled with δ. The states of the probabilistic executions are

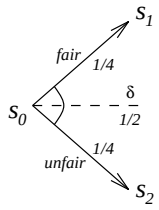


Figure 4-7: A probabilistic execution of the triggered coin flipper.

not represented as finite execution fragments since their position in the diagrams gives enough information. Similarly, we omit writing explicitly all the transitions that lead to $\mathcal{D}(\delta)$ (e.g., states s_1 and s_2 in Figure 4-7).

We now have enough structure to understand better the role of δ. In ordinary automata a scheduler has the possibility not to schedule anything at any point, leading to a finite execution. Such assumption is meaningful if the actions enabled from a given state model some input that comes from the external environment. In the probabilistic framework it is also possible to schedule no transition from some point. Since a scheduler may use randomization in its choices, it is also possible that from some specific state nothing is scheduled only with some probability p, say $1/2$.

Example 4.2.4 (The role of δ) In the triggered coin flipper of Figure 4-5 a user can flip a fair coin to decide whether to push a button, and then, if the coin flip yields head, flip another coin to decide which button to press. In the transition that leaves from s_0 we need some structure that represents the fact that nothing is scheduled from s_0 with probability $1/2$: we use δ for this purpose. Figure 4-7 represents the probabilistic execution that we have just described. ■

Since a probabilistic execution fragment is itself a probabilistic automaton, it is possible to talk about the executions of a probabilistic execution fragment, that is, the ways in which the probabilistic choices can be resolved in a probabilistic execution fragment. However, since at any point q it is possible not to schedule anything, if we want to be able to study the probabilistic behavior of a probabilistic execution fragment then we need to distinguish between being in q with the possibility to proceed and being in q without any possibility to proceed. For example, in the probabilistic execution of Figure 4-7 we need to distinguish between being in s_0 before performing the transition enabled from s_0 and being in s_0 after performing the transition. We represent this second condition by writing $s_0\delta$. In general, we introduce a notion of an extended execution fragment, which is used in Section 4.2.5 to study the probability space associated with a probabilistic execution.

An *extended execution* (fragment) of a probabilistic automaton M, denoted by α, is either an execution (fragment) of M, or a sequence $\alpha'\delta$, where α' is a finite execution (fragment) of M. The sequences $s_0\delta$ and $s_0\, fair\, s_1\delta$ are examples of extended executions of the probabilistic execution of Figure 4-7.

There is a close relationship between the extended executions of a probabilistic automaton and the extended executions of one of its probabilistic execution fragments. Here we define two operators that make such a relationship explicit. Let M be a probabilistic automaton and

let H be a probabilistic execution fragment of M. Let q_0 be the start state of H. For each extended execution $\alpha = q_0 a_1 q_1 \cdots$ of H, let

$$\alpha\downarrow \stackrel{\triangle}{=} \begin{cases} q_0 \frown lstate(q_0) a_1 lstate(q_1) a_2 \cdots & \text{if } \alpha \text{ does not end in } \delta, \\ q_0 \frown lstate(q_0) a_1 lstate(q_1) a_2 \cdots a_n lstate(q_n)\delta & \text{if } \alpha = q_0 a_1 q_1 \cdots a_n q_n \delta. \end{cases} \tag{4.2}$$

It is immediate to observe that $\alpha\downarrow$ is an extended execution fragment of M. For each extended execution fragment α of M such that $q_0 \leq \alpha$, i.e., $\alpha = q_0 \frown s_0 a_1 s_1 \cdots$, let

$$\alpha\uparrow q_0 \stackrel{\triangle}{=} \begin{cases} q_0 a_1 (q_0 \frown s_0 a_1 s_1) a_2 (q_0 \frown s_0 a_1 s_1 a_2 s_2) \cdots & \text{if } \alpha \text{ does not end in } \delta, \\ q_0 a_1 (q_0 \frown s_0 a_1 s_1) \cdots (q_0 \frown s_0 a_1 s_1 \cdots a_n s_n)\delta & \text{if } \alpha = q_0 \frown s_0 a_1 s_1 \cdots a_n s_n \delta. \end{cases} \tag{4.3}$$

It is immediate to observe that $\alpha\uparrow q_0$ is an extended execution of some probabilistic execution fragment of M. Moreover, the following proposition holds.

Proposition 4.2.2 *Let H be a probabilistic execution fragment of a probabilistic automaton M, and let q_0 be the start state of H. Then, for each extended execution α of H,*

$$(\alpha\downarrow)\uparrow q_0 = \alpha, \tag{4.4}$$

and for each extended execution fragment α of M starting with q_0,

$$(\alpha\uparrow q_0)\downarrow = \alpha. \tag{4.5}$$

Proof. Simple analysis of the definitions. ■

The bottom line is that it is possible to talk about extended executions of H by analyzing only extended execution fragments of M.

4.2.4 Notational Conventions

For the analysis of probabilistic automata and of probabilistic executions we need to refer to explicit objects like transitions or probability spaces associated with transitions. In this section we give a collection of notational conventions that ease the identification of each object.

Transitions

We denote a generic transition of a probabilistic automaton by tr, possibly decorated with primes and indices. For each transition $tr = (s, \mathcal{P})$, we denote $\mathcal{P}$ alternatively by $\mathcal{P}_{tr}$. If tr is a simple transition, represented by $(s, a, \mathcal{P})$, we abuse notation by denoting $\mathcal{P}$ by $\mathcal{P}_{tr}$ as well. The context will always clarify the probability space that we denote. If $(s, \mathcal{P})$ is a transition, we use any set of actions V to denote the event $\{(a, s') \in \Omega \mid a \in V\}$ that expresses the occurrence of an action from V in $\mathcal{P}$, and we use any set of states U to denote the event $\{(a, s') \in \Omega \mid s' \in U\}$ that expresses the occurrence of a state from U in $\mathcal{P}$. We drop the set notation for singletons. Thus, $P[a]$ is the probability that action a occurs in the transition $(s, \mathcal{P})$.

If M is a fully probabilistic automaton and s is a state of M, then we denote the unique transition enabled from s in M by tr_s^M, and we denote the probability space that appears in tr_s^M by $\mathcal{P}_s^M$. Thus, $tr_s^M = (s, \mathcal{P}_s^M)$. We drop M from the notation whenever it is clear from the context. This notation is important to handle probabilistic execution fragments.

Transition Prefixing and Suffixing

Throughout the thesis we use transitions of probabilistic automata and transitions of probabilistic execution fragments interchangeably. If H is a probabilistic execution fragment of a probabilistic automaton M, then there is a strong relation between the transitions of H and some of the combined transitions of M. We exploit such a correspondence through two operations on transitions. The first operation is called transition prefixing and adds some partial history to the states of a transition; the second operation is called transition suffixing and removes some partial history from the states of a transition. These operations are used mainly in the proofs of the results of this thesis.

Let $tr = (s, \mathcal{P})$ be a combined transition of a probabilistic automaton M, and let α be a finite execution fragment of M such that $lstate(\alpha) = s$. Then the transition $\alpha \frown tr$ is defined to be $(\alpha, \alpha \frown \mathcal{P})$. We call the operation $\alpha\frown$ *transition prefixing*.

Let $tr = (q, \mathcal{P})$ be a transition of a probabilistic execution fragment H, and let $q' \leq q$. Let $\triangleright q'$ be a function that applied to a pair (a, q'') of Ω returns $(a, q'' \triangleright q')$, and applied to δ returns δ. Let $\mathcal{P} \triangleright q'$ denote the result of applying $\triangleright q'$ to $\mathcal{P}$. Then the transition $tr \triangleright q'$ is defined to be $(q \triangleright q', \mathcal{P} \triangleright q')$. We call the operation $\triangleright q'$ *transition suffixing*.

The following properties concern distributivity of transition prefixing and suffixing with respect to combination of transitions.

Proposition 4.2.3 *Let M be a probabilistic automaton, and let q be a finite execution fragment of M.*

1. $q \frown \sum_i p_i tr_i = \sum_i p_i (q \frown tr_i)$, *where each tr_i is a transition of M.*
2. $\sum_i p_i tr_i \triangleright q = \sum_i p_i (tr_i \triangleright q)$, *where each tr_i is a transition of some probabilistic execution fragment of M.*

Proof. Simple manipulation of the definitions. ■

4.2.5 Events

At this point we need to define formally how to compute the probability of some event in a probabilistic execution. Although it is intuitively simple to understand the probability of a finite execution to occur, it is not as intuitive to understand how to deal with arbitrary properties. A probabilistic execution can be countably branching, and can have uncountably many executions. As an example, consider a probabilistic execution that at any point draws a natural number $n > 0$ with probability $1/2^n$. What is measurable? What is the probability of a generic event?

In this section we define a suitable probability space for a generic probabilistic execution fragment H of a probabilistic automaton M. Specifically, given a probabilistic execution fragment H we define a probability space $\mathcal{P}_H$ as the completion of another probability space $\mathcal{P}'_H$ which is defined as follows. Define an extended execution α of H to be *complete* iff either α is infinite or $\alpha = \alpha'\delta$ and $\delta \in \Omega^H_{lstate(\alpha')}$. Then, the sample space Ω'_H is the set of extended executions of M that originate from complete extended executions of H, i.e.,

$$\Omega'_H \triangleq \{\alpha\downarrow \mid \alpha \text{ is a complete extended execution of } H\}. \tag{4.6}$$

The occurrence of a finite extended execution α of M can be expressed by the set

$$C_\alpha^H \triangleq \{\alpha' \in \Omega'_H \mid \alpha \leq \alpha'\}, \tag{4.7}$$

called a *cone*. We drop H from C_α^H whenever it is clear from the context. Let $\mathcal{C}_H$ be the set of cones of H. Then define $\mathcal{F}'_H$ to be the σ-field generated by $\mathcal{C}_H$, i.e.,

$$\mathcal{F}'_H \triangleq \sigma(\mathcal{C}_H). \tag{4.8}$$

To define a probability measure on $\mathcal{F}'_H$, we start by defining a measure μ_H on $\mathcal{C}_H$ such that $\mu_H(\Omega_H) = 1$. Then we show that μ_H can be extended uniquely to a measure $\bar{\mu}_H$ on $F(\mathcal{C}_H)$, where $F(\mathcal{C}_H)$ is built according to Proposition 3.1.1. Finally we use the extension theorem (Theorem 3.1.2) to show that μ_H can be extended uniquely to a probability measure P'_H on $\sigma(F(\mathcal{C}_H)) = \sigma(\mathcal{C}_H)$.

The measure $\mu_H(C_\alpha^H)$ of a cone C_α^H is the product of the probabilities associated with each edge that generates α in H. Formally, let q_0 be the start state of H. If $\alpha \leq q_0$, then

$$\mu_H(C_\alpha^H) \triangleq 1; \tag{4.9}$$

if $\alpha = q_0 \frown s_0 a_1 s_1 \cdots s_{n-1} a_n s_n$, then

$$\mu_H(C_\alpha^H) \triangleq P_{q_0}^H[(a_1, q_1)] \cdots P_{q_{n-1}}^H[(a_n, q_n)], \tag{4.10}$$

where for each i, $1 \leq i < n$, $q_i = q_0 \frown s_0 a_1 s_1 \cdots s_{i-1} a_i s_i$; if $\alpha = q_0 \frown s_0 a_1 s_1 \cdots s_{n-1} a_n s_n \delta$, then

$$\mu_H(C_\alpha^H) \triangleq P_{q_0}^H[(a_1, q_1)] \cdots P_{q_{n-1}}^H[(a_n, q_n)] P_{q_n}[\delta], \tag{4.11}$$

where for each i, $1 \leq i \leq n$, $q_i = q_0 \frown s_0 a_1 s_1 \cdots s_{i-1} a_i s_i$.

Example 4.2.5 (Some commonly used events) Before proving that the construction of $\mathcal{P}'_H$ is correct, we give some examples of events. The set describing the occurrence of an action a (eventually a occurs) can be expressed as a union of cones of the form C_α such that a appears in α. Moreover, any union of cones can be described as a union of disjoint cones (follows from Lemma 4.2.4 below). Since a probabilistic execution fragment is at most countably branching, the number of distinct cones in $\mathcal{C}_H$ is at most countable, and thus the occurrence of a can be expressed as a countable union of disjoint cones, i.e., it is an event of $\mathcal{F}'_H$. More generally, any arbitrary union of cones is an event. We call such events *finitely satisfiable*. The reason for the word "satisfiable" is that it is possible to determine whether an execution α of Ω'_H is within a finitely satisfiable event by observing just a finite prefix of α. That finite prefix is sufficient to determine that the property represented by the given event is satisfied.

The set describing the non-occurrence of an action a is also an event, since it is the complement of a finitely satisfiable event. Similarly, the occurrence, or non-occurrence, of any finite sequence of actions is an event. For each natural number n, the occurrence of exactly n a's is an event: it is the intersection of the event expressing the occurrence of at least n a's and the event expressing the non-occurrence of $n+1$ a's. Finally, the occurrence of infinitely many a's is an event: it is the countable intersection of the events expressing the occurrence of at least i a's, $i \geq 0$. ■

We now move to the proof that $\mathcal{P}'_H$ is well defined. First we use ordinal induction to show that the function μ_H defined on $\mathcal{C}_H$ is σ-additive, and thus that μ_H is a measure on $\mathcal{C}_H$ (Lemma 4.2.6); then we show that there is a unique extension of μ_H to $F(\mathcal{C}_H)$ (Lemmas 4.2.7, 4.2.8, and 4.2.9). Finally, we use the extension theorem to conclude that P'_H is well defined.

Lemma 4.2.4 *Let $C_{\alpha_1}, C_{\alpha_2} \in \Omega_H$. If $\alpha_1 \leq \alpha_2$ then $C_{\alpha_1} \subseteq C_{\alpha_2}$. If $\alpha_1 \not\leq \alpha_2$ and $\alpha_2 \not\leq \alpha_1$ then $C_{\alpha_1} \cap C_{\alpha_2} = \emptyset$.*

Proof. Simple analysis of the definitions. ■

Lemma 4.2.5 *Let H be a probabilistic execution of a probabilistic automaton M, and let q be a state of H. Suppose that there is a transition enabled from q in H. Then*

$$\mu_H(C_q) = \begin{cases} \sum_{(a,q') \in \Omega_q^H} \mu_H(C_{q'}) & \text{if } \delta \notin \Omega_q^H \\ \sum_{(a,q') \in \Omega_q^H} \mu_H(C_{q'}) + \mu_H(C_{q\delta}) & \text{if } \delta \in \Omega_q^H. \end{cases} \tag{4.12}$$

Proof. Simple analysis of the definitions. ■

Lemma 4.2.6 *The function μ_H is σ-additive on $\mathcal{C}_H$, and $\mu_H(\Omega_H) = 1$.*

Proof. By definition $\mu_H(\Omega'_H) = 1$, hence it is sufficient to show σ-additivity. Let q be an extended execution of M, and let Θ be a set of incomparable extended executions of M such that $C_q = \cup_{q' \in \Theta} C_{q'}$. If q ends in δ, then Θ contains only one element and σ-additivity is trivially satisfied. Thus, assume that q does not end in δ, and hence q is a state of H, and that Θ contains at least two elements. From Lemma 4.2.4, q is a prefix of each extended execution of Θ. For each state q' of H, let $\Theta_{q'}$ be the set $\{q'' \in \Theta \mid q' \leq q''\}$. We show σ-additivity in two steps: first we assign an ordinal depth to some of the states of H and we show that q is assigned a depth; then we show that $\mu_H(C_q) = \sum_{q' \in \Theta} \mu_H(C_{q'})$ by ordinal induction on the depth assigned to q.

The depth of each state q' within some cone $C_{q''}$ ($q'' \leq q'$), where $q'' \in \Theta$, is 0, and the depth of each state q' with no successors is 0. For each other state q' such that each of its successors has a depth, if $\{depth(q'') \mid \exists_a (a, q'') \in \Omega_{q'}^H\}$ has a maximum, then

$$depth(q') = max(\{depth(q'') \mid \exists_a (a, q'') \in \Omega_{q'}^H\}) + 1, \tag{4.13}$$

otherwise, if $\{depth(q'') \mid \exists_a (a, q'') \in \Omega_{q'}\}$ does not have a maximum, then

$$depth(q') = sup(\{depth(q'') \mid \exists_a (a, q'') \in \Omega_{q'}^H\}). \tag{4.14}$$

Consider a maximal assignment to the states of H, i.e., an assignment that cannot be extended using the rules above, and suppose by contradiction that q is not assigned a depth. Then consider the following sequence of states of H. Let $q_0 = q$, and, for each $i > 0$, let q_i be a state of H such that $(a_i, q_i) \in \Omega_{q_{i-1}}$, and q_i is not assigned a depth. For each i, the state q_i exists since otherwise, if there exists an i such that for each $(a_i, q_i) \in \Omega_{q_{i-1}}$, q_i is assigned a depth, then q_{i-1} would be assigned a depth. Note that the q_i's form a chain under prefix ordering, i.e., for each i, j, if $i \leq j$ then $q_i \leq q_j$. Consider the execution $\alpha_\infty = \lim_i q_i$. From its definition, α_∞ is an execution of C_q. Then, from hypothesis, α_∞ is an execution of $\cup_{q' \in \Theta} C_{q'}$, and therefore α_∞ is an execution of some $C_{q'}$ such that $q' \in \Theta$. By definition of a cone, q' is a prefix of α_∞.

Thus, $q' = q_k$ for some $k \geq 0$. But then q_k is within the cone $C_{q'}$, and thus it is assigned depth 0. This contradicts the fact that q_k is not assigned any depth.

Let γ be the ordinal depth assigned to q. We show that $\mu_H(C_q) = \sum_{q' \in \Theta} \mu_H(C_{q'})$ by ordinal induction on γ. If $\gamma = 0$, then Θ is either $\{q\}$ or $\{q\delta\}$, and the result is trivial. Let γ be a successor ordinal or a limit ordinal. From Lemma 4.2.5, $\mu_H(C_q) = \sum_{(a,q') \in \Omega_q} \mu_H(C_{q'})$ if $\delta \notin \Omega_q$, and $\mu_H(C_q) = \sum_{(a,q') \in \Omega_q} \mu_H(C_{q'}) + \mu_H(C_{q\delta})$ if $\delta \in \Omega_q$. For each $(a,q') \in \Omega_q$, $C_{q'} = \cup_{q'' \in \Theta_{q'}} C_{q''}$. Moreover, for each $(a,q') \in \Omega_q$, the depth of q' is less than γ. By induction, $\mu_H(C_{q'}) = \sum_{q'' \in \Theta_{q'}} \mu_H(C_{q''})$. Thus, $\mu_H(C_q) = \sum_{(a,q') \in \Omega_q} \sum_{q'' \in \Theta_{q'}} \mu_H(C_{q''}) = \sum_{q' \in \Theta} \mu_H(C_{q'})$ if $\delta \notin \Omega_q$, and $\mu_H(C_q) = \sum_{(a,q') \in \Omega_q} \sum_{q'' \in \Theta_{q'}} \mu_H(C_{q''}) + \mu_H(C_{q\delta}) = \sum_{q' \in \Theta} \mu_H(C_{q'})$ if $\delta \in \Omega_q$. ■

Lemma 4.2.7 *There exists a unique extension μ'_H of μ_H to $F_1(\mathcal{C}_H)$.*

Proof. There is a unique way to extend the measure of the cones to their complements since for each α, $\mu_H(C_\alpha) + \mu_H(\Omega_H - C_\alpha) = 1$. Therefore μ'_H coincides with μ_H on the cones and is defined to be $1 - \mu_H(C_\alpha)$ for the complement of any cone C_α. Since, by the countably branching structure of H, the complement of a cone is a countable union of cones, σ-additivity is preserved. ■

Lemma 4.2.8 *There exists a unique extension μ''_H of μ'_H to $F_2(\mathcal{C}_H)$.*

Proof. The intersection of finitely many sets of $F_1(\mathcal{C}_H)$ is a countable union of cones. Therefore σ-additivity enforces a unique measure on the new sets of $F_1(\mathcal{C}_H)$. ■

Lemma 4.2.9 *There exists a unique extension μ'''_H of μ''_H to $F_3(\mathcal{C}_H)$.*

Proof. There is a unique way of assigning a measure to the finite union of disjoint sets whose measure is known, i.e., adding up their measures. Since all the sets of $F_3(\mathcal{C}_H)$ are countable unions of cones, σ-additivity is preserved. ■

Theorem 4.2.10 *There exists a unique extension P'_H of μ_H to the σ-algebra $\mathcal{F}'_H$.*

Proof. By Theorem 3.1.2, define P'_H to be the unique extension of μ'''_H to $\mathcal{F}'_H$. ■

4.2.6 Finite Probabilistic Executions, Prefixes, Conditionals, and Suffixes

We extend the notions of finiteness, prefix and suffix to the probabilistic framework. Here we add also a notion of conditional probabilistic execution which is not meaningful in the non-probabilistic case and which plays a crucial role in some of the proofs of Chapter 5.

Finite Probabilistic Executions

Informally, finiteness means that the tree representation of a probabilistic execution fragment has a finite depth. Thus, a probabilistic execution fragment H is *finite* iff there exists a natural number n such that the length of each state of H is at most n.

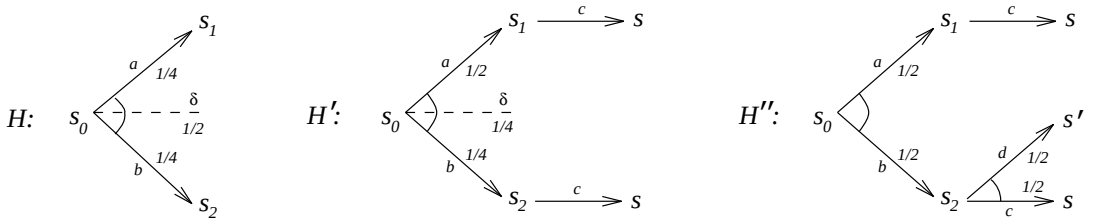


Figure 4-8: Examples of the prefix relation.

Prefixes

The idea of a prefix of a probabilistic execution fragment is more complicated than the definition of prefix for ordinary automata. To get a better understanding of the problem, consider the definition of prefix for ordinary execution fragments: $\alpha \leq \alpha'$ iff either $\alpha = \alpha'$, or α is finite and there is an execution fragment α'' such that $\alpha' = \alpha^\frown\alpha''$. Another way to interpret this definition is to observe that if α is finite, then there is exactly one point in α, which we call a *point of extension*, from which nothing is scheduled, and in that case α' is obtained by extending α from its unique point of extension. With the word "extending" we mean "adding transitions". In other words, an execution fragment α is a prefix of an execution fragment α' iff α' is obtained from α by adding transitions, possibly none, from all the points of extension of α, i.e., from all the points of α where nothing is scheduled. We apply the same observation to probabilistic execution fragments, where a point of extension is any point where δ occurs.

Example 4.2.6 (Prefixes) Consider the probabilistic execution fragment H of Figure 4-8. It is easy to see that s_1 and s_2 are points of extension in H. However, also s_0 is a point of extension since in H nothing is scheduled from s_0 with probability $1/2$. The probabilistic execution fragment H' of Figure 4-8 is an extension of H. States s_1 and s_2 are extended with transitions labeled with c, and half of the extendible part of s_0 is extended with the transition $s_0 \xrightarrow{a} s_1$, i.e., we have added the transition $(s_0, \mathcal{U}((a, s_1), \delta))$ to the extendible part of s_0. Since the extension from s_0 overlaps with one of the edges leaving s_0 in H, the effect that we observe in H' is that s_1 is reached with a higher probability.

Consider now the probabilistic execution fragment H'' of Figure 4-8. H'' is an extension of H', but this time something counterintuitive has happened; namely, the edge labeled with action c that leaves from state s_2 has a lower probability in H'' than in H'. The reason for this difference is that the extendible part of s_0 is extended with a transition $s_0 \xrightarrow{b} s_2$ followed by $s_2 \xrightarrow{c} s'$. Thus, half of the transition leaving from s_2 in H'' is due to the previous behavior of H', and half of the transition leaving from s_2 in H'' is due to the extension from s_0. However, the probability of the cone $C_{s_0 b s_2 c s}$ is the same in H' and in H''. ■

A formal definition of a prefix works as follows. A probabilistic execution fragment H is a prefix of a probabilistic execution fragment H', denoted by $H \leq H'$, iff

1. H and H' have the same start state, and
2. for each state q of H, $P_H[C_q] \leq P_{H'}[C_q]$.

Observe that the definition of a prefix for ordinary executions is a special case of the definition we have just given.

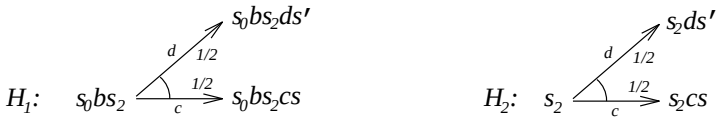


Figure 4-9: Conditionals and suffixes.

Conditionals

Let H be a probabilistic execution fragment of a probabilistic automaton M, and let q be either a state of H or a prefix of the start state of H. We want to identify the part of H that describes what happens conditional to the occurrence of q. The new structure, which we denote by $H|q$, is a new probabilistic execution fragment defined as follows:

1. $states(H|q) = \{q' \in states(H) \mid q \leq q'\}$;
2. $start(H|q) = min(states(H|q))$, where the minimum is taken under prefix ordering,
3. for each state q' of $H|q$, $tr^{H|q}_{q'} = tr^{H}_{q'}$.

$H|q$ is called a *conditional* probabilistic execution fragment.

Example 4.2.7 (Conditionals) The probabilistic execution fragment H_1 of Figure 4-9 is an example of a conditional probabilistic execution fragment. Specifically, $H_1 = H''|(s_0as_2)$, where H'' is represented in Figure 4-8. In Figure 4-9 we represent explicitly the states of H_1 for clarity. The conditional operation essentially extracts the subtree of H'' that starts with s_0as_2. ■

It is easy to check that $(\Omega_{H|q}, \mathcal{F}_{H|q}, P_{H|q})$ and $(\Omega_H|C_q, \mathcal{F}_H|C_q, P_H|C_q)$ are the same probability space (cf. Section 3.1.8). Indeed, the sample sets are the same, the generators are the same, and the probability measures coincide on the generators. Thus, the following proposition, which is used in Chapter 5, is true.

Proposition 4.2.11 *Let H be a probabilistic execution fragment of a probabilistic automaton M, and let q be either a state of H, or a prefix of the start state of H. Then, for each subset E of $\Omega_{H|q}$,*

1. *$E \in \mathcal{F}_{H|q}$ iff $E \in \mathcal{F}_H$.*
2. *If E is an event, then $P_H[E] = P_H[C_q]P_{H|q}[E]$.* ■

Suffixes

The definition of a suffix is similar to the definition of a conditional; the difference is that in the definition of $H \triangleright q$ we drop q from each state of H, i.e., we forget part of the past history. Formally, let H be a probabilistic execution fragment of a probabilistic automaton M, and let q be either a state of H or a prefix of the start state of H. Then $H \triangleright q$ is a new probabilistic execution fragment defined as follows:

1. $states(H \triangleright q) = \{q' \triangleright q \mid q' \in states(H), q \leq q'\}$,

2. $start(H \triangleright q) = min(states(H \triangleright q))$, where the minimum is taken under prefix ordering,

3. for each state q' of H', $tr_{q'}^{H \triangleright q} = tr_{q \frown q'}^{H} \triangleright q$.

$H \triangleright q$ is called a *suffix* of H. It is a simple inductive argument to show that $H \triangleright q$ is indeed a probabilistic execution fragment of M. Observe that the definition of a suffix for ordinary executions is a special case of the definition we have just given.

Example 4.2.8 (Suffixes) The probabilistic execution fragment H_2 of Figure 4-9 is an example of a suffix. Specifically, $H_2 = H'' \triangleright (s_0 a s_2)$, where H'' is represented in Figure 4-8. The suffixing operation essentially extracts the subtree of H'' that starts with $s_0 a s_2$ and removes from each state the prefix $s_0 a s_2$. ■

It is easy to check that the probability spaces $\mathcal{P}_{H \triangleright q}$ and $\mathcal{P}_{H|q}$ are in a one-to-one correspondence through the measurable function $f : \Omega_{H \triangleright q} \rightarrow \Omega_{H|q}$ such that for each $\alpha \in \Omega_{H \triangleright q}$, $f(\alpha) = q \frown \alpha$. The inverse of f is also measurable and associates $\alpha \triangleright q$ with each execution α of $\Omega_{H|q}$. Thus, directly from Proposition 4.2.11, we get the following proposition.

Proposition 4.2.12 *Let H be a probabilistic execution fragment of a probabilistic automaton M, and let q be either a state of H, or a prefix of the start state of H. Then, for each subset E of $\Omega_{H \triangleright q}$,*

1. *$E \in \mathcal{F}_{H \triangleright q}$ iff $(q \frown E) \in \mathcal{F}_H$.*

2. *If E is an event, then $P_H[q \frown E] = P_H[C_q] P_{H \triangleright q}[E]$.* ■

4.2.7 Notation for Transitions

In this section we extend the arrow notation for transitions that is used for ordinary automata. The extension that we present is meaningful for simple transitions only.

An alternative representation for a simple transition $(s, a, \mathcal{P})$ of a probabilistic automaton M is $s \xrightarrow{a} \mathcal{P}$. Thus, differently from the non-probabilistic case, a transition leads to a distribution over states. If $\mathcal{P}$ is a Dirac distribution, say $\mathcal{D}(s')$, then we can represent the corresponding transition by $s \xrightarrow{a} s'$. Thus, the notation for ordinary automata becomes a special case of the notation for probabilistic automata. If $(s, a, \mathcal{P})$ is a simple combined transition of M, then we represent the transition alternatively by $s \xrightarrow{a}_C \mathcal{P}$, where the letter C stands for "combined".

The extension of weak transitions is more complicated. The expression $s \stackrel{a}{\Longrightarrow} \mathcal{P}$ means that $\mathcal{P}$ is reached from s through a sequence of transitions of M, some of which are internal. The main difference from the non-probabilistic case is that in the probabilistic framework the transitions involved form a tree rather than a linear chain. Formally, $s \stackrel{a}{\Longrightarrow} \mathcal{P}$, where a is either an external action or the empty sequence and $\mathcal{P}$ is a probability distribution over states, iff there is a probabilistic execution fragment H such that

1. the start state of H is s;

2. $P_H[\{\alpha\delta \mid \alpha\delta \in \Omega_H\}] = 1$, i.e., the probability of termination in H is 1;

3. for each $\alpha\delta \in \Omega_H$, $trace(\alpha) = a$;

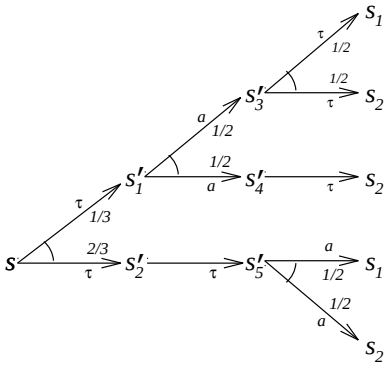


Figure 4-10: A representation of a weak transition with action a.

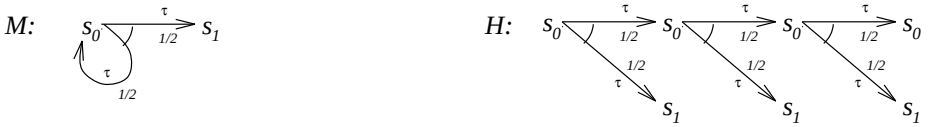


Figure 4-11: A weak transition of a probabilistic automaton with cycles.

4. $\mathcal{P} = \mathit{lstate}(\delta\text{-}\mathit{strip}(\mathcal{P}_H))$, where $\delta\text{-}\mathit{strip}(\mathcal{P}_H)$ is the probability space $\mathcal{P}'$ such that $\Omega' = \{\alpha \mid \alpha\delta \in \Omega_H\}$, and for each $\alpha \in \Omega'$, $P'[\alpha] = P_H[C_{\alpha\delta}]$;

5. for each state q of H, either tr_q^H is the pair $(\mathit{lstate}(q), \mathcal{D}(\delta))$, or the transition that corresponds to tr_q^H is a transition of M.

A weak combined transition, $s \stackrel{a}{\Longrightarrow}_{\mathrm{C}} \mathcal{P}$, is defined as a weak transition by dropping Condition 5. Throughout the thesis we also the extend the function $\delta\text{-}\mathit{strip}$ to extended execution fragment; its action is to remove the symbol δ at the end of each extended execution fragment.

Example 4.2.9 (Weak transitions) Figure 4-10 represents a weak transition with action a that leads to state s_1 with probability $5/12$ and to state s_2 with probability $7/12$. The action τ represents any internal action. From the formal definition of a weak transition, a tree that represents a weak transition may have an infinite branching structure, i.e., it may have transitions that lead to countably many states, and may have some infinite paths; however, the set of infinite paths has probability 0.

Figure 4-11 represents a weak transition of a probabilistic automaton with cycles in its transition relation. Specifically, H represents the weak transition $s_0 \Longrightarrow \mathcal{P}$, where $P[s_0] = 1/8$ and $P[s_1] = 7/8$. If we extend H indefinitely on its right, then we obtain a new probabilistic execution fragment that represents the weak transition $s_0 \Longrightarrow \mathcal{D}(s_1)$. Observe that the new probabilistic execution fragment has an infinite path that occurs with probability 0. Furthermore, observe that there is no other way to reach state s_1 with probability 1. ■

Remark 4.2.10 According to our definition, a weak transition can be obtained by concatenating together infinitely many transitions of a probabilistic automaton. A reasonable objection to this definition is that sometimes scheduling infinitely many transitions is unfeasible. In the

timed framework this problem is even more important since it is feasible to assume that there is some limit to the number of transitions that can be scheduled in a finite time. Thus, a more reasonable and intuitive definition of a weak transition would require the probabilistic execution fragment H that represent a weak transition not to have any infinite path. All the results that we prove in this thesis are valid for the more general definition where H can have infinite paths as well as for the stricter definition where H does not have any infinite path. Therefore, we use the more general definition throughout. The reader is free to think of the simpler definition to get a better intuition of what happens. ■

An alternative way to represent a weak transition, which is used to prove the theorems of Chapter 8, is by means of a *generator*. If H represents a weak combined transition, then a generator can be seen as an object that chooses the combined transitions of M that lead to H (in Chapter 5 this object is also called an adversary). More precisely, a generator is a function $\mathcal{O}$ that associates a weak combined transition of M with each finite execution fragment of M. Before stating the formal properties that a generator satisfies, we give an example of the generator for the weak transition of Figure 4-10.

Example 4.2.11 (Generators) Recall from Section 3.1.10 that $\mathcal{U}(x, y)$ denotes the probability space that assigns x and y probability 1/2 each. Then, the generator for the weak transition of Figure 4-10 is the function $\mathcal{O}$ where

$$\begin{array}{lll} & & \mathcal{O}(s\tau s_1' a s_3') = (s_3', \tau, \mathcal{U}(s_1, s_2)) \\ & \mathcal{O}(s\tau s_1') = (s_1', a, \mathcal{U}(s_3', s_4')) & \mathcal{O}(s\tau s_1' a s_4') = (s_4', \tau, \mathcal{D}(s_2)) \\ \mathcal{O}(s) = (s, \tau, \mathcal{U}(s_1', s_2')) & \mathcal{O}(s\tau s_2') = (s_2', \tau, \mathcal{D}(s_5')) & \mathcal{O}(s\tau s_2' \tau s_5') = (s_5', a, \mathcal{U}(s_1, s_2)) \end{array}$$

and $\mathcal{O}(\alpha) = (lstate(\alpha), \mathcal{D}(\delta))$ for each α that is not considered above. The layout of the definition above reflects the shape of the probabilistic execution fragment of Figure 4-10.

Thus, if we denote the probabilistic execution fragment of Figure 4-10 by H, $\mathcal{O}$ is the function that for each state q of H gives the combined transition of M that corresponds to tr_q^H. Function $\mathcal{O}$ is also minimal in the sense that it returns a transition different from $(lstate(q), \mathcal{D}(\delta))$ only from those states q that are relevant for the construction of H. We call *active* all the states of H that enable some transition; we call *reachable* all the reachable states of H; we call *terminal* all the states q of H such that $\delta \in \Omega_q^H$. ■

Let M be a probabilistic automaton and let s be a state of M. A generator for a weak (combined) transition $s \stackrel{a \restriction ext(M)}{\Longrightarrow} \mathcal{P}$ of M is a function $\mathcal{O}$ that associates a (combined) transition of M with each finite execution fragment of M such that the following conditions are satisfied.

1. If $\mathcal{O}(\alpha) = (s', \mathcal{P})$, then $s' = lstate(\alpha)$. Call α *active* if $\mathcal{P} \neq \mathcal{D}(\delta)$.
2. If $\alpha b s'$ is active, then $fstate(\alpha) = s$ and $(b, s') \in \Omega_{\mathcal{O}(\alpha)}$.
3. Call α *reachable* iff either $\alpha = s$ or $\alpha = \alpha' b s'$ and $(b, s') \in \Omega_{\mathcal{O}(\alpha')}$. Call α *terminal* iff α is reachable and $P_{\mathcal{O}(\alpha a s')}[\delta] > 0$. Then, for each terminal α, the trace of α is $a \restriction ext(M)$.
4. For each reachable execution fragment $\alpha = s a_1 s_1 a_2 s_2 \cdots a_k s_k$, let

$$P_\alpha^{\mathcal{O}} \stackrel{\triangle}{=} \prod_{0 \leq i < k} P_{\mathcal{O}(s a_1 s_1 \cdots a_i s_i)}[(a_{i+1} s_{i+1})],$$

Then,

$$\Omega = \{lstate(\alpha) \mid terminal(\alpha)\},$$

and for each $s' \in \Omega$,

$$P[s'] = \sum_{\alpha | lstate(\alpha)=s', terminal(\alpha)} P_\alpha^{\mathcal{O}} P_{\mathcal{O}(\alpha)}[\delta].$$

Condition 1 says that the transition that $\mathcal{O}(\alpha)$ returns is a legal transition of M from $lstate(\alpha)$; Condition 2 guarantees that the active execution fragments are exactly those that are relevant for the weak transition denoted by $\mathcal{O}$; Condition 3 ensures that the weak transition represented by $\mathcal{O}$ has action $a \upharpoonright ext(M)$; Condition 4 computes the probability space reached in the transition represented by $\mathcal{O}$, which must coincide with $\mathcal{P}$. The term $P_\alpha^{\mathcal{O}}$ represents the probability of performing α if $\mathcal{O}$ resolves the nondeterminism in M. Observe that terminal execution fragments must be reachable with probability 1 if we want the structure computed in Condition 4 to be a probability space.

Proposition 4.2.13 *There is a weak combined transition $s \stackrel{a}{\Longrightarrow} \mathcal{P}$ of M iff there is a function $\mathcal{O}$ that satisfies the five conditions of the definition of a generator.*

Proof. Simple analysis of the definitions. ■

4.3 Parallel Composition

In this section we extend to the probabilistic framework the parallel composition operator and the notion of a projection of ordinary automata. The parallel composition of simple probabilistic automata can be defined easily by enforcing synchronization on the common actions as in the non-probabilistic case; for general probabilistic automata, however, it is not clear how to give a synchronization rule. We discuss the problems involved at the end of the section.

4.3.1 Parallel Composition of Simple Probabilistic Automata

Two probabilistic automata M_1 and M_2 are *compatible* iff

$$int(M_1) \cap acts(M_2) = \emptyset \text{ and } acts(M_1) \cap int(M_2) = \emptyset.$$

The *parallel composition* of two compatible simple probabilistic automata M_1 and M_2, denoted by $M_1 \| M_2$, is the simple probabilistic automaton M such that

1. $states(M) = states(M_1) \times states(M_2)$.
2. $start(M) = start(M_1) \times start(M_2)$.
3. $sig(M) = (ext(M_1) \cup ext(M_2), int(M_1) \cup int(M_2))$.
4. $((s_1, s_2), a, \mathcal{P}) \in trans(M)$ iff $\mathcal{P} = \mathcal{P}_1 \otimes \mathcal{P}_2$ where

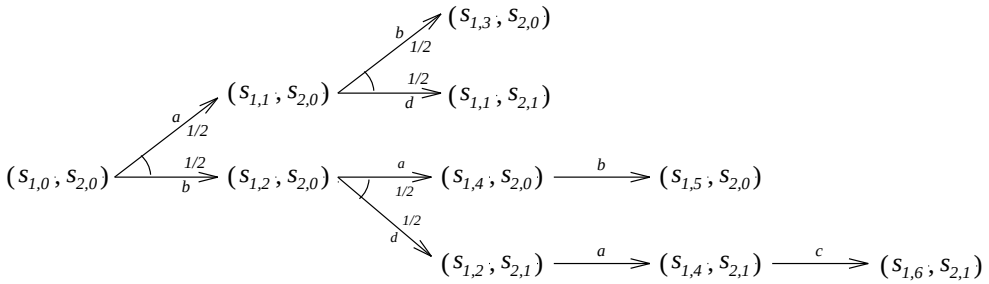


Figure 4-12: A probabilistic execution fragment of $M_1 \| M_2$.

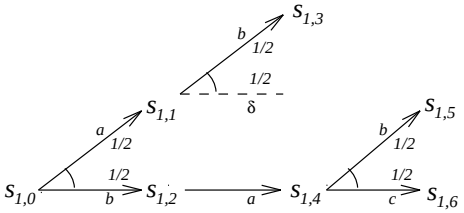


Figure 4-13: The projection onto M_1 of the probabilistic execution fragment of Figure 4-10.

(a) if $a \in acts(M_1)$ then $(s_1, a, \mathcal{P}_1) \in trans(M_1)$, else $\mathcal{P}_1 = \mathcal{D}(s_1)$, and
(b) if $a \in acts(M_2)$ then $(s_2, a, \mathcal{P}_2) \in trans(M_2)$, else $\mathcal{P}_2 = \mathcal{D}(s_2)$.

Similar to the non-probabilistic case, two simple probabilistic automata synchronize on their common actions and evolve independently on the others. Whenever a synchronization occurs, the state that is reached is obtained by choosing a state independently for each of the probabilistic automata involved.

4.3.2 Projection of Probabilistic Executions

The Structure of the Problem

Let $M = M_1 \| M_2$, and let H be a probabilistic execution fragment of M. We want to determine the view that M_1 has of H, or, in other words, what probabilistic execution M_1 performs in order for $M_1 \| M_2$ to produce H. To understand the complexity of the problem, consider the probabilistic execution fragment of Figure 4-12, and consider its projection onto M_1, represented in Figure 4-13. Actions a, b and c are actions of M_1, while action d is an action of M_2. Thus, there is no communication between M_1 and M_2. Denote the probabilistic execution fragment of Figure 4-12 by H, and denote the probabilistic execution fragment of Figure 4-13 by H_1. The projections of the states are ordinary projections of pairs onto their first component. The transitions, however, are harder to understand. We analyze them one by one.

$s_{1,0}$ The transition leaving $s_{1,0}$ is obtained directly from the transition leaving $(s_{1,0}, s_{2,0})$ in H by projecting onto M_1 the target states.

$s_{1,2}$ The transition leaving $s_{1,2}$ is obtained by combining the transitions leaving states $(s_{1,2}, s_{2,0})$ and $(s_{1,2}, s_{2,1})$, each one with probability $1/2$. The two transitions leaving $(s_{1,2}, s_{2,0})$ and

$(s_{1,2}, s_{2,1})$ have the same projection onto M_1, and thus the transition leaving $s_{1,2}$ in H_1 is $s_{1,2} \xrightarrow{a} s_{1,4}$. From the point of view of M_1, there is just a transition $s_{1,2} \xrightarrow{a} s_{1,4}$; nothing is visible about the behavior of M_2.

To give a better idea of what we mean by "visible", suppose that M_1 is a student who has to write a report and suppose that the report can be written using a pen (action c) or using a pencil (action b). Suppose that the teacher may be able to get a pencil eraser (action d) and possibly erase the report written by the student once it is ready for grading. Then the scheduler is an arbiter who gives the student a pen if the teacher gets an eraser. If the student starts in state $s_{1,2}$, then from the point of view of the student the material for the report is prepared (action a), and then the arbiter gives the student a pen with probability 1/2 and a pencil with probability 1/2; nothing is known about the time the the arbiter made the choice and the reason for which the choice was made. We can also think of the student as being alone in a room and the arbiter as being a person who brings to the student either a pen or a pencil once the material for the report is ready.

The detailed computation of the transition leaving from $s_{1,2}$ in H_1 works as follows: we start from state $(s_{1,2}, s_{2,0})$, which is the first state reached in H where M_1 is in $s_{1,2}$, and we analyze its outgoing edges. We include directly all the edges labeled with actions of M_1 in the transition leaving $s_{1,2}$; for the other edges, we move to the states that they lead to, in our case $(s_{1,2}, s_{2,1})$, and we repeat the same procedure keeping in mind that the probability of the new edges must be multiplied by the probability of reaching the state under consideration. Thus, the edge labeled with a that leaves $(s_{1,2}, s_{2,0})$ is given probability 1/2 since its probability is 1/2, and the edge that leaves $(s_{1,2}, s_{2,1})$ is given probability 1/2 since the probability of reaching $(s_{1,2}, s_{2,1})$ from $(s_{1,2}, s_{2,0})$ is 1/2.

$s_{1,4}$ For the transition leaving $s_{1,4}$, we observe that in H there are two states, namely $(s_{1,4}, s_{2,0})$ and $(s_{1,4}, s_{2,1})$, that can be reached separately and whose first component is $s_{1,4}$. Each one of the two states is reached in H with probability 1/4. The difference between the case for state $s_{1,2}$ and this case is that in the case for $s_{1,2}$ state $(s_{1,2}, s_{2,0})$ occurs before $(s_{1,2}, s_{2,1})$, while in this case there is no relationship between the occurrences of $(s_{1,4}, s_{2,0})$, and $(s_{1,4}, s_{2,1})$. The transition leaving $s_{1,4}$ depends on the state of M_2 which, conditional on M_1 being in $s_{1,4}$, is 1/2 for $s_{2,0}$ and 1/2 for $s_{2,1}$. Thus, from the point of view of M_1, since the state of M_2 is unknown, there is a transition from $s_{1,4}$ that with probability 1/2 leads to the occurrence of action b and with probability 1/2 leads to the occurrence of action c. Essentially we have normalized to 1 the probabilities of states $(s_{1,4}, s_{2,0})$ and $(s_{1,4}, s_{2,1})$ before considering their effect on M_1.

$s_{1,1}$ The transition leaving $s_{1,1}$ shows why we need the symbol δ in the transitions of a probabilistic automaton. From state $(s_{1,1}, s_{2,0})$ there is a transition where action b occurs with probability 1/2 and action τ occurs with probability 1/2. After τ is performed, nothing is scheduled. Thus, from the point of view of M_1, nothing is scheduled from $s_{1,1}$ with probability 1/2; the transition of M_2 is not visible by M_1.

Action Restricted Transitions

The formal definition of a projection relies on a new operation on transitions, called action restriction, which is used also in several other parts of the thesis. The action restriction operation allows us to consider only those edges of a transition that are labeled with actions from a designated set V. For example, V could be the set of actions of a specific probabilistic automaton.

Formally, let M be a probabilistic automaton, V be a set of actions of M, and $tr = (s, \mathcal{P})$ be a transition of M. The transition tr restricted to actions from V, denoted by $tr \upharpoonright V$, is the pair $(s, \mathcal{P}')$ where $\mathcal{P}'$ is obtained from $\mathcal{P}$ by considering only the edges labeled with actions from V and by normalizing their probability to 1, i.e.,

- $\Omega' = \begin{cases} \{(a, s') \in \Omega \mid a \in V\} & \text{if } P[V] > 0 \\ \{\delta\} & \text{otherwise} \end{cases}$
- if $P[V] > 0$, then for each $(a, s') \in \Omega'$, $P'[(a, s')] = P[(a, s')]/P[V]$.

Two properties of action restriction concern commutativity with transition prefixing, and distributivity with respect to combination of transitions. These properties are used in the proofs of other important results of this thesis. The reader may skip the formal statements for the moment and refer back to them when they are used.

Proposition 4.3.1 *For each q and tr such that one of the expressions below is defined,*

$$q \frown (tr \upharpoonright V) = (q \frown tr) \upharpoonright V.$$

Proof. Simple manipulation of the definitions. ■

Proposition 4.3.2 *Let $\{t_i\}_{i \in I}$ be a collection of transitions leaving from a given state s, and let $\{p_i\}_{i \in I}$ be a collection of real numbers between 0 and 1 such that $\sum_{i \in I} p_i \leq 1$. Let V be a set of actions. Then*

$$(\sum_i p_i tr_i) \upharpoonright V = \sum_i \frac{p_i P_{tr_i}[V]}{\sum_i p_i P_{tr_i}[V]} (tr_i \upharpoonright V),$$

where we use the convention that $0/0 = 0$.

Proof. Let

$$(s, \mathcal{P}) \triangleq \sum_i p_i tr_i, \tag{4.15}$$

$$(s, \mathcal{P}') \triangleq (\sum_i p_i tr_i) \upharpoonright V, \tag{4.16}$$

$$(s, \mathcal{P}'') \triangleq \sum_i \frac{p_i P_{tr_i}[V]}{\sum_i p_i P_{tr_i}[V]} (tr_i \upharpoonright V). \tag{4.17}$$

We need to show that $\mathcal{P}'$ and $\mathcal{P}''$ are the same probability space.

If $P[V] = 0$, then both $\mathcal{P}'$ and $\mathcal{P}''$ are $\mathcal{D}(\delta)$ and we are done. Otherwise, observe that neither Ω' nor Ω'' contain δ. Consider any pair (a, s'). Then,

	$(a, s') \in \Omega'$	
iff	$(a, s') \in \Omega$ and $a \in V$	from (4.16) and (4.15)
iff	$\exists_i (a, s') \in \Omega_{tr_i}, p_i > 0$, and $a \in V$	from (4.15)
iff	$\exists_i (a, s') \in \Omega_{tr_i \upharpoonright V}$ and $p_i > 0$	from the definition of $tr_i \upharpoonright V$
iff	$(a, s') \in \Omega''$	from (4.17).

Consider now a pair (a, s') of Ω'. From the definition of action restriction and (4.16),

$$P'[(a, s')] = P[(a, s')]/P[V]. \tag{4.18}$$

From the definition of $\mathcal{P}$ (Equation (4.15)), the right side of Equation 4.18 can be rewritten into

$$\sum_i \frac{p_i}{\sum_i p_i P_{tr_i}[V]} P_{tr_i}[(a, s')], \tag{4.19}$$

where $\sum_i p_i P_{tr_i}[V]$ is an alternative expression of $P[V]$ that follows directly from (4.16). By multiplying and dividing each i^{th} summand of Expression 4.19 by $P_{tr_i}[V]$, we obtain

$$\sum_i \frac{p_i P_{tr_i}[V]}{\sum_i p_i P_{tr_i}[V]} (P_{tr_i}[(a, s')]/P_{tr_i}[V]). \tag{4.20}$$

Since $P_{tr_i}[(a, s')]/P_{tr_i}[V] = P_{tr_i \upharpoonright V}[(a, s')]$, from the definition of $\mathcal{P}''$ (Equation (4.17)), Expression 4.20 can be rewritten into $P''[(a, s')]$. Thus, $P'[(a, s')] = P''[(a, s')]$. This is enough to show that $\mathcal{P}' = \mathcal{P}''$. ■

Definition of Projection

We give first the formal definition of a projection, and then we illustrate its critical parts by analyzing the example of Figures 4-12 and 4-13. It is very important to understand Expressions (4.21) and (4.22) since similar expressions will be used in several other parts of the thesis without any further explanation except for formal proofs.

Let $M = M_1 \| M_2$, and let H be a probabilistic execution fragment of M.

Let $tr = (q, \mathcal{P})$ be an action restricted transition of H such that only actions of M_i, $i = 1, 2$, appear in tr. Define the projection operator on the elements of Ω as follows: $(a, q')\lceil M_i = (a, q'\lceil M_i)$, and $\delta \lceil M_i = \delta$. Recall from Section 3.1.5 that the projection can be extended to discrete probability spaces. The projection of tr onto M_i, denoted by $tr\lceil M_i$, is the pair $(q\lceil M_i, \mathcal{P}\lceil M_i)$.

The projection of H onto M_i, denoted by $H\lceil M_i$, is the fully probabilistic automaton H' such that

1. $states(H') = \{q\lceil M_i \mid q \in states(H)\}$;
2. $start(H') = \{q\lceil M_i \mid q \in start(H)\}$;
3. $sig(H') = sig(M_i)$;

4. for each state q of H', let $q\rceil H$ be the set of states of H that projected onto M_i give q, and let $min(q\rceil H)$ be the set of minimal states of $q\rceil H$ under prefix ordering. For each $q' \in (q\rceil H)$, let

$$\bar{p}_{q'}^{q\rceil H} \triangleq \frac{P_H[C_{\bar{q}}]}{\sum_{q'' \in min(q\rceil H)} P_H[C_{q''}]}. \tag{4.21}$$

The transition enabled from q in H' is

$$tr_q^{H'} \triangleq \sum_{q' \in q\rceil H} \bar{p}_{q'}^{q\rceil H} P_{q'}^H[acts(M_i)](tr_{q'}^H \upharpoonright acts(M_i))\lceil M_i. \tag{4.22}$$

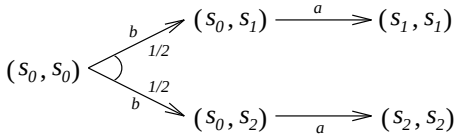
Each summand of Expression 4.22 corresponds to the analysis of one of the states of H that can influence the transition enabled from q in H'. The subexpression $(tr_{q'}^H \upharpoonright acts(M_i))\lceil M_i$ selects the part of the transition leaving from q' where M_i is active, and projects onto M_i the target states of the selected part; the subexpression $\bar{p}_{q'}^{q\rceil H} P_{q'}^H[acts(M_i)]$ expresses the probability with which q' influences the transition enabled from q. $P_{q'}^H[acts(M_i)]$ is the probability that $tr_{q'}^H$ does something visible by M_i, and $\bar{p}_{q'}^{q\rceil H}$ is the probability of being in q' conditional on M_i being in q. Its value is given by Expression 4.21 and can be understood as follows. The state q' is either a minimal state of $q\rceil H$ or is reached from a minimal state through a sequence of edges with actions not in $acts(M_i)$. The probability of being in q', conditional on M_i being in q, is the normalized probability of being in the minimal state of $q\rceil H$ that precedes q' multiplied by the probability of reaching q' from that minimal state. We encourage the reader to apply Expression (4.22) to the states $s_{1,0}, s_{1,1}, s_{1,2}$, and $s_{1,4}$ of Figure 4-13 to familiarize with the definition. As examples, observe that $min((s_{1,0}bs_{1,2})\rceil H) = \{(s_{1,0}, s_{2,0})b(s_{1,2}, s_{2,0})\}$ and that $min((s_{1,0}bs_{1,2}as_{1,4})\rceil H) = \{(s_{1,0}, s_{2,0})b(s_{1,2}, s_{2,0})a(s_{1,4}, s_{2,0}), (s_{1,0}, s_{2,0})b(s_{1,2}, s_{2,0})d(s_{1,2}, s_{2,1})a(s_{1,4}, s_{2,1})\}$.

If we analyze the state $s_{1,3}$ of Figure 4-13 and we use Expression 4.22 to compute the transition leaving $s_{1,3}$, then we discover that the sum of the probabilities involved is not 1. This is because there is a part of the transition leaving $(s_{1,3}, s_{2,0})$ where no action of M_1 ever occurs. From the point of view of M_1 nothing is scheduled; this is the reason of our choice of deadlock by default in the definition of the combination of transitions (cf. Section 4.2.2).

We now move to Proposition 4.3.4, which is the equivalent of Proposition 3.2.1 for the probabilistic framework. Specifically, we show that the projection of a probabilistic execution fragment H of $M_1 \| M_2$ onto one of its components M_i is a probabilistic execution fragment of M_i. Proposition 3.2.1 is important because it shows that every computation of a parallel composition is the result of some computation of each of the components. One of the reasons for our use of randomized schedulers in the model is to make sure that Proposition 3.2.1 is valid. Before proving this result, we show that its converse does not hold, i.e., that there are structures that look like a probabilistic execution, that projected onto each component give a probabilistic execution of a component, but that are not probabilistic executions themselves.

Example 4.3.1 (Failure of the converse of Proposition 4.3.4) Consider the probabilistic automata of Figure 4-14.a, and consider a potential probabilistic execution of the composition as represented in Figure 4-14.b. Denote the two probabilistic automata of Figure 4-14.a by M_1 and M_2, and denote the structure of Figure 4-14.b by H. The projections of H onto M_1 and

a) Two compatible simple probabilistic automata.



b) A potential probabilistic execution of the composition.

Figure 4-14: A counterexample to the converse of the projection proposition.

M_2 give a probabilistic execution of M_1 and M_2, respectively. The diagrams of Figure 4-14.a can be viewed as the projections of H as well. However, H is not a probabilistic execution of $M_1 \| M_2$ since in no place of M_1 it is possible to have a Dirac transition to s_1 or s_2. ■

The rest of this section is dedicated to the proof of the proposition that corresponds to Proposition 3.2.1 and to the proof of an additional result (Proposition 4.3.5) that gives a meaning to the denominator of Expression (4.21). We first state two preliminary properties of projection of transitions (Proposition 4.3.3).

Proposition 4.3.3 *Let $M = M_1 \| M_2$. Then, for $i = 1, 2$,*

1. $(\sum_j p_j tr_j) \lceil M_i = \sum_j p_j (tr_j \lceil M_i)$.
2. $(q \frown tr) \lceil M_i = (q \lceil M_i) \frown tr \lceil M_i$.

Proof. Simple manipulation of the definitions. ■

Proposition 4.3.4 *Let $M = M_1 \| M_2$, and let H be a probabilistic execution fragment of M. Then $H \lceil M_1 \in prexec(M_1)$ and $H \lceil M_2 \in prexec(M_2)$.*

Proof. We show that $H \lceil M_1 \in prexec(M_1)$; the other statement follows from a symmetric argument. Let H_1 denote $H \lceil M_1$. From Proposition 3.2.1, the states of H_1 are execution fragments of M_1.

Consider now a state q of H_1. We need to show that there is a combined transition tr of M_1 that corresponds to $tr_q^{H_1}$, i.e., such that $tr_q^{H_1} = q \frown tr$. From Propositions 4.2.1 and 4.2.3, it is sufficient to show that for each state q' of $q \rceil H$, there is a combined transition $tr(q')$ of M_1 such that

$$(tr_{q'}^{H} \upharpoonright acts(M_1)) \lceil M_1 = q \frown tr(q'). \tag{4.23}$$

Then, the transition tr would be

$$tr = \sum_{q' \in q\rceil H} \bar{p}_{q'}^{q\rceil H} P_{q'}^{H}[acts(M_i)]tr(q'). \tag{4.24}$$

Proposition 4.2.1 is used to show that tr is a combined transition of M_1; Proposition 4.2.3 is used to show that $q \frown tr = tr_q^{H_1}$. Since H is a probabilistic execution fragment of M, for each state q' of $q\rceil H$ there exists a combined transition $tr'(q')$ of M such that

$$tr_{q'}^{H} = q' \frown tr'(q'). \tag{4.25}$$

From the definition of a combined transition, there is a collection of transitions $\{tr'(q', i)\}_{i \in I}$ of M, and a collection of probabilities $\{p_i\}_{i \in I}$, such that

$$tr'(q') = \sum_i p_i tr'(q', i). \tag{4.26}$$

Note that each transition $tr'(q', i)$ is a simple transition. From the definition of action restriction and (4.26), there is a subset J of I, and a collection of non-zero probabilities $\{p'_j\}_{j \in J}$, such that

$$tr'(q') \upharpoonright acts(M_1) = \sum_j p'_j tr'(q', j). \tag{4.27}$$

If we apply transition prefix with q' to both sides of Equation 4.27, we use commutativity of action restriction with respect to transition prefixing (Proposition 4.3.1) and (4.25) on the left expression, and we use distributivity of transition prefixing with respect to combination of transitions (Proposition 4.2.3) on the right expression, then we obtain

$$tr_{q'}^{H} \upharpoonright acts(M_1) = \sum_j p'_j \left(q' \frown tr'(q', j)\right). \tag{4.28}$$

By projecting buth sides of (4.28) onto M_1, and using distributivity of projection with respect to combination of transitions (Proposition 4.3.3) and commutativity of projection and transition prefixing (Proposition 4.3.3) on the right expression, we obtain

$$(tr_{q'}^{H} \upharpoonright acts(M_1))\lceil M_1 = \sum_j p'_j \left(q \frown (tr'(q', j)\lceil M_1)\right). \tag{4.29}$$

From the distributivity of transition prefixing with respect to combination of transitions (Proposition 4.2.3), Equation 4.29 becomes

$$(tr_{q'}^{H} \upharpoonright acts(M_1))\lceil M_1 = q \frown \sum_j p'_j (tr'(q', j)\lceil M_1). \tag{4.30}$$

From standard properties of the projection of product probability distributions (cf. Section 3.1.6) and the definition of parallel composition, each $tr'(q', j)\lceil M_1$ is a transition of M_1. Thus, $\sum_j p'_j tr'(q', j)\lceil M_1$ is the combined transition of M_1 that satisfies Equation 4.23.

Finally, we need to show that each state q of H_1 is reachable. This is shown by induction on the length of q, where the base case is the start state of H_1. The start state of H_1 is trivially reachable. Consider a state qas of H_1. By induction, q is reachable. Let q' be a minimal state of $(qas)\rceil H$. Then, $q' = q''a(s, s_2)$, where q'' is a state of $q\rceil H$ and s_2 is a state

of M_2. Moreover, $(a, q') \in \Omega_{tr^H_{q''}}$, and thus, $(a, qas) \in \Omega_{(tr^H_{q''} \upharpoonright acts(M_1))\lceil M_1}$. Since no edges with probability 0 are allowed in a probabilistic automaton, the term $\bar{p}^{q\rceil H}_{q''} P^H_{q''}[acts(M_i)]$ is not 0, and thus $(a, qas) \in \Omega^{H_1}_q$. This means that qas is reachable. ■

We conclude this section with another property of projections that gives a meaning to the denominator of Expression (4.21). Specifically, the proposition below allows us to compute the probability of a finitely satisfiable event of the projection of a probabilistic execution fragment H by computing the probability of a finitely satisfiable event of H. Observe that the right expression of (4.31) is indeed the denominator of (4.21).

Proposition 4.3.5 *Let $M = M_1 \| M_2$, and let H be a probabilistic execution fragment of M. Let H_i be $H \lceil M_i$, $i = 1, 2$. Let q be a state of H_i. Then,*

$$P_{H_i}[C_q] = \sum_{q' \in min(q\rceil H)} P_H[C_{q'}]. \tag{4.31}$$

Proof. The proof is by induction on the length of q, where the base case is for the start state of H_i. If q is the start state of H_i, then the start state of H is the only minimal state of $q\rceil H$. Both the cones denoted by the two states have probability 1.

Consider now the case for qas. From the definition of the probability of a cone,

$$P_{H_1}[C_{qas}] = P_{H_1}[C_q] P^{H_1}_q[(a, qas)]. \tag{4.32}$$

By using Expression 4.22 and the definitions of action restriction and projection, the term $P^{H_1}_q[(a, qas)]$ can be rewritten into

$$\sum_{q' \in q\rceil H} \bar{p}^{q\rceil H}_{q'} P^H_{q'}[acts(M_i)] \left(\sum_{q'' \in (qas)\rceil H | (a,q'') \in \Omega^H_{q'}} P^H_{q'}[(a, q'')] / P^H_{q'}[acts(M_i)] \right), \tag{4.33}$$

which becomes

$$\sum_{q' \in q\rceil H} \bar{p}^{q\rceil H}_{q'} \left(\sum_{q'' \in (qas)\rceil H | (a,q'') \in \Omega^H_{q'}} P^H_{q'}[(a, q'')] \right), \tag{4.34}$$

after simplifying the term $P^H_{q'}[acts(M_i)]$. The case when $P^H_{q'}[acts(M_i)] = 0$ is not a problem since the innermost sum of Expression 4.33 would be empty. By expanding $\bar{p}^{q\rceil H}_{q'}$ in Expression 4.34 with its definition (Equation 4.21), applying induction to $P_{H_1}[C_q]$ in Expression 4.32, and simplifying algebraically, Equation 4.32 can be rewritten into

$$P_{H_1}[C_{qas}] = \sum_{q' \in q\rceil H} \sum_{q'' \in (qas)\rceil H | (a,q'') \in \Omega^H_{q'}} P_H[C_{q'}] P^H_{q'}[(a, q'')]. \tag{4.35}$$

Indeed, the denominator of the expansion of $\bar{p}^{q\rceil H}_{q'}$ coincides with the expansion of $P_{H_1}[C_q]$.

From the definition of the probability of a cone, the terms $P_H[C_{q'}] P^H_{q'}[(a, q'')]$ that appear in Equation 4.35 can be rewritten into $P_H[C_{q''}]$.

Consider now one of the states q'' of the right side of Equation 4.35. Then $q''\lceil M_i = qas$, and there exists a state q' of $q\rceil H$ such that $(a, q'') \in \Omega_{q'}$. This means that q'' can be expressed as $q'as'$ for some state s' of M. Since $q'\lceil M_i = q$, then q'' is a minimal state of $(qas)\rceil H$. Conversely, let q'' be a minimal state of $(qas)\rceil H$. Then q'' can be expressed as $q'as'$ for some state q' of H and some state s' of M (otherwise q'' would not be minimal). Moreover, q' is a state of $q\rceil H$ and $(a, q'') \in \Omega^H_{q'}$. Thus, q'' is considered in Equation 4.35. Finally, each minimal state q'' of $(qas)\rceil H$ is considered at most once in Equation 4.35, since there is at most one state q' in H such that $(a, q'') \in \Omega^H_{q'}$. Thus, Equation 4.35 can be rewritten into

$$P_{H_1}[C_{qas}] = \sum_{q'' \in min((qas)\rceil H)} P_H[C_{q''}], \tag{4.36}$$

which is what we needed to show. ■

4.3.3 Parallel Composition for General Probabilistic Automata

In this section we give an idea of the problems that arise in defining parallel composition for general probabilistic automata. The discussion is rather informal: we want to give just an idea of why our intuition does not work in this case.

The main problem that needs to be addressed is to choose when two transitions should synchronize and how the synchronization would occur. We analyze the problem through some toy examples. Consider two probabilistic automata M_1, M_2 with no internal actions and such that $ext(M_1) = \{a, b, c, d\}$ and $ext(M_2) = \{a, b, c, e\}$. Let (s_1, s_2) be a reachable state of $M_1 \| M_2$, and consider the following cases.

1. Suppose that from state s_1 of M_1 there is a transition tr_1 giving actions a, b probability $1/2$ to occur, and suppose that from state s_2 of M_2 there is a transition tr_2 giving actions a, b probability $1/2$ to occur.

   If we choose not to synchronize tr_1 and tr_2, then the only transitions that can be synchronized are the simple transitions, leading to a trivial parallel composition operator that does not handle any kind of transition with probabilistic choices over actions. The transitions tr_1 and tr_2 cannot be scheduled even independently, since otherwise the CSP synchronization style would be violated.

   If we choose to synchronize tr_1 and tr_2, then both M_1 and M_2 choose an action between a and b. If the actions coincide, then there is a synchronization, otherwise we have two possible choices in our definition: either the system deadlocks, or the random draws are repeated. The first approach coincides with viewing each probabilistic automaton as deciding its next action probabilistically independently of the other interacting automaton; the second approach is the one outlined in [GSST90], where essentially deadlock is not allowed, and assumes some dependence between the involved probabilistic automata.

   For the rest of the discussion we assume that the transitions tr_1 and tr_2 do synchronize; however, we leave unspecified the way in which tr_1 and tr_2 synchronize.

2. Suppose that from state s_1 of M_1 there is a transition tr_1 giving actions a, b probability $1/2$ to occur, and suppose that from state s_2 of M_2 there is a transition tr_2 giving actions a, c probability $1/2$ to occur.

Note that actions a, b and c are all in common between M_1 and M_2. If we choose not to synchronize tr_1 and tr_2, then only transitions involving the same sets of actions can synchronize. However, we have the same problem outlined in Case 1, where neither tr_1, nor tr_2 can be scheduled independently.

If we choose to synchronize tr_1 and tr_2, then, since a is the only action that is in common between tr_1 and tr_2, the only action that can occur is a. Its probability is either 1 or $1/4$ depending on how the synchronization in Case 1 is resolved. However, in both cases the only action that appears in the sample space of the composite transition is a.

For the rest of the discussion we assume that the transitions tr_1 and tr_2 do synchronize. Once again, we leave unspecified the way in which tr_1 and tr_2 synchronize.

3. Suppose that from state s_1 of M_1 there is a transition tr_1 giving actions a, b, d probability $1/3$ to occur, and suppose that from state s_2 of M_2 there is a transition tr_2 giving actions a, b, e probability $1/3$ to occur.

In this case each transition has some actions that are in common between M_1 and M_2, and some actions that are not in common.

If we choose not to synchronize tr_1 and tr_2, then, beside the fact that tr_1 and tr_2 could not be scheduled independently, the parallel composition operator would not be associative. Consider two new probabilistic automata M_1', M_2' with the same actions as M_1 and M_2, respectively. Suppose that from state s_1' of M_1' there is a transition tr_1' giving actions a, b probability $1/2$ to occur, and suppose that from state s_2' of M_2' there is a transition tr_2' giving actions a, b probability $1/2$ to occur.

If we consider $(M_1' \| M_1) \| (M_2 \| M_2')$, then in state $((s_1', s_1), (s_2, s_2'))$ tr_1 would synchronize with tr_1' leading to a transition that involves actions a and b only, tr_2 would synchronize with tr_2' leading to a transition that involves actions a and b only, and the two new

transitions would synchronize because of Case 1, leading to a transition that involves actions a and b. If we consider $(M_1' \| (M_1 \| M_2)) \| M_2'$, then in state $((s_1', (s_1, s_2)), s_2')$ tr_1 and tr_2 would not synchronize, and thus associativity is broken.

If we choose to synchronize tr_1 and tr_2, then problems arise due to the presence of actions that are not in common between M_1 and M_2. In particular we do not know what to do if M_1 draws action d and M_2 draws action e, or if M_1 draws action d and M_2 draws action a. Since we do not want to assume anything about the respective probabilistic behaviors of M_1 and M_2, at least the first case is an evident case of nondeterminism.

However, even by dealing with the first case above by means of nondeterminism, only one of actions d, e can be performed. Suppose that d is chosen, and thus M_1 performs a transition while M_2 does not. What happens to M_2? Is action e supposed to be chosen already after d is performed? Otherwise, what is the probability for e to occur? At this point we do not see any choice that would coincide with any reasonable intuition about the involved systems.

In the second case we are sure that action a cannot occur. Does this mean that action d occurs for sure? Or does this mean that a deadlock can occur? With what probabilities? Once again, intuition does not help in this case.

The main problem, which is evident especially from Case 3, is that we do not know who is in control of a system, and thus, whenever there is a conflict that is not solved by nondeterminism alone, we do not know what probability distribution to use to resolve the conflict. However, if we decorate probabilistic automata with some additional structure that clarifies who is in control of what actions [LT87], then parallel composition can be extended safely to some forms of general probabilistic automata, where the external actions are partitioned into *input* and *output* actions, the transitions that contain some input action are simple transitions, and input actions are enabled from every state (cf. Section 13.2.2). An observation along this line appears in [WSS94].

4.4 Other Useful Operators

There are two other operators on probabilistic automata that should be mentioned, since they are used in general on ordinary automata. In this section we provide a short description of those operators. Since the relative theory is simple, this is the only point where we mention these operators during the development of the probabilistic model.

4.4.1 Action Renaming

Let ρ be a one-to-one function whose domain is $acts(M)$. Define $Rename_\rho(M)$ to be the probabilistic automaton M' such that

1. $states(M') = states(M)$.
2. $start(M') = start(M)$.
3. $sig(M') = (\rho(ext(M)), \rho(int(M)))$.

4. $(s, \mathcal{P}) \in trans(M')$ iff there exists a transition $(s, \mathcal{P}')$ of M such that $\mathcal{P} = \rho'(\mathcal{P}')$, where $\rho'((a, s')) = (\rho(a), s')$ for each $(a, s') \in \Omega'$, and $\rho'(\delta) = \delta$.

Thus, the effect of $Rename_\rho$ is to change the action names of M. The restriction on ρ to be one-to-one can be relaxed as long as internal and external actions are not mixed, i.e., there is no pair of actions a, b where a is an external action, b is an internal action, and $\rho(a) = \rho(b)$.

4.4.2 Action Hiding

Let M be a probabilistic automaton, and let I be a set of actions. Then $Hide_I(M)$ is defined to be a probabilistic automaton M' that is the same as M, except that

$$sig(M') = (ext(M) - I, int(M) \cup I).$$

That is, the actions in the set I are hidden from the external environment.

4.5 Discussion

The generative model of probabilistic processes of van Glabbeek et al. [GSST90] is a special case of a fully probabilistic automaton; simple probabilistic automata are partially captured by the reactive model of [GSST90] in the sense that the reactive model assumes some form of nondeterminism between different actions. However, the reactive model does not allow nondeterministic choices between transitions involving the same action. By restricting simple probabilistic automata to have finitely many states, we obtain objects with a structure similar to that of the Concurrent Labeled Markov Chains of [Han91]; however, in our model we do not need to distinguish between nondeterministic and probabilistic states. In our model nondeterminism is obtained by means of the structure of the transition relation. This allows us to retain most of the traditional notation that is used for automata.

Our parallel composition operator is defined only for simple probabilistic automata, and thus a natural objection is that after all we are dealing just with the reactive model. Furthermore, the reactive model is the least general according to [GSST90]. Although we recognize that our simple probabilistic automata constitute a restricted model and that it would be desirable to extend the parallel composition operator to general probabilistic automata, we do not think that it is possible to use the classification of [GSST90] to judge the expressivity of simple probabilistic automata. The classification of [GSST90] is based on a synchronous parallel composition, while our parallel composition is based on a conservative extension of the parallel composition of CSP [Hoa85]. Furthermore, in the classification of [GSST90] a model is more general if it contains less nondeterminism, while in our model nondeterminism is one of the key features.

Chapter 5

Direct Verification: Stating a Property

This chapter presents a method to study the properties that a probabilistic automaton satisfies. We describe how an informally stated property can be made rigorous, and we show how simple statements can be combined together to give more complex statements. In Chapter 6 we develop techniques to prove from scratch that a probabilistic automaton satisfies a given property.

Part of this chapter is based on discussion with Isaac Saias who provided us with the motivations for the definition of progress statements (Section 5.5) and for the statement of the concatenation theorem (Theorem 5.5.2).

5.1 The Method of Analysis

If we read through the papers on randomized algorithms and we look at the statements of correctness, we see claims like

> "*Whenever the algorithm X starts in a condition Y, no matter what the adversary does, the algorithm X achieves the goal Z with probability at least p.*"

For convenience, denote the statement above by S. A possible concrete instantiation of S is the following:

> "*Consider a distributed system X, composed of n processors, that provides services under request and suppose that some request R comes. Then, independently of the relative order in which the n processors complete their operations (no matter what the adversary does), a response to R is given eventually (the goal Z) with probability at least $2/3$.*

Let us try to understand the meaning of the statement S. First of all, in S there is an entity, called *adversary*, that affects the performance of algorithm X. The adversary is seen as a malicious entity that degrades the performance of X as much as possible.

If X is a distributed algorithm that runs on n separate processes, then the adversary is the entity that chooses what process performs the next transition, and possibly what the external environment does. To account for all the possible ways to schedule processes, the adversary

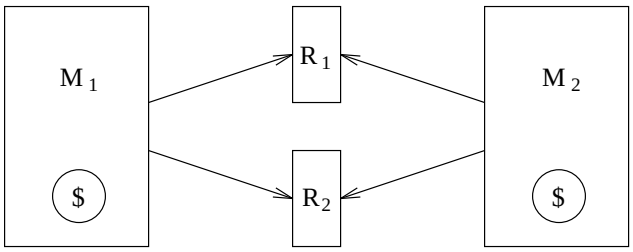


Figure 5-1: A toy resource allocation protocol.

bases its choices on a complete knowledge of the state of a system, including its past history. If the algorithm is represented as a probabilistic automaton, then an adversary is the object that resolves the nondeterminism. In other words, an adversary is a scheduler seen as a malicious entity.

However, not all the schedulers guarantee in general that some specific property is satisfied. For example, an adversary is usually required to be fair to all the processes of a system in order to guarantee progress. In other cases, an adversary is not allowed to base its choices on a complete knowledge of the history of a system: the correctness of an algorithm may rely on the adversary not to use the results of previous random draws in choosing the next process to be scheduled. Thus, in the statement S there is usually an implicit assumption that an adversary has some limitations.

Example 5.1.1 (A toy resource allocation protocol) Figure 5-1 illustrates a toy scenario where correctness is guaranteed only for adversaries that do not know the outcome of the random draws of the processes. Two processes M_1 and M_2 compete for two resources R_1 and R_2. Each process continuously runs through the following cycle:

1. flip a coin to choose a resource;
2. if the chosen resource is free, then get it;
3. if you hold the resource, then return it.

That is, each process continuously tries to get a randomly chosen resource and then returns it, possibly after using the resource. Of course this is a stupid protocol, but it highlights several aspects of randomized distributed algorithms. Suppose every adversary to be fair, meaning that both processes perform infinitely many transitions. A malicious adversary can create a situation where M_1 never succeeds in obtaining a resource with an arbitrarily high probability. The adversary works as follows. Fix an arbitrary probability p such that $0 < p < 1$, and consider a collection of probabilities $\{p_i\}_{i \in N}$ such that $\prod_i p_i = p$. We know that such a collection of probabilities exists. Then the adversary works in rounds, where at round i the following happens:

a. M_1 is scheduled until it flips its coin;

b. M_2 is scheduled for sufficiently many times so that it gets the resource chosen by M_1 with probability at least p_i (finitely many times are sufficient). As soon as M_2 gets the resource chosen by M_1 the control goes to c;

c. M_1 is scheduled to check its resource and fails to get it.

In this case M_1 fails to obtain a resource with probability at least p. On the other hand, if an adversary is not allowed to base its choices on the outcome of the coin flips, or better, if an adversary chooses the next process that performs a transition based only on the order in which processes were scheduled in the past, then each process eventually gets a resource with probability 1 (this fact is proved in Section 6.6). Such an adversary is called an *oblivious adversary* or an *off-line scheduler*. ■

Let us move back to the problem of understanding the statement S. Consider a valid adversary $\mathcal{A}$, i.e., an adversary that satisfies the limitations that are implicitly assumed for S. Let M be a probabilistic automaton that describes algorithm X, and consider an arbitrary starting point q for M, i.e., q is a finite execution fragment of M that describes a partial evolution of M. If we let $\mathcal{A}$ resolve the nondeterminism in M starting from the knowledge that q occurred, then we obtain a probabilistic execution fragment of M, which we denote by $prexec(M, \mathcal{A}, q)$. According to S, if q satisfies condition Y, then $prexec(M, \mathcal{A}, q)$ should satisfy property Z with probability at least p. However, Z is a property of M, and not a property of $prexec(M, \mathcal{A}, q)$. Thus, we need a way to associate with $prexec(M, \mathcal{A}, q)$ the event that expresses Z. The object that does this operation is called an *event schema*. At this point it is possible to formalize S by stating the following:

> "*For each valid adversary $\mathcal{A}$ and each valid starting condition q, the probability of the event associated with $prexec(M, \mathcal{A}, q)$ is at least p.*"

This is an example of what we call a *probabilistic statement*.

A probabilistic statement that plays an important role in our analysis is denoted by

$$U \xrightarrow[p]{} _{Advs} U', \tag{5.1}$$

where U and U' are sets of states, p is a probability, and $Advs$ is a set of adversaries. We call such a statement a *progress statement*. Its meaning is that if a protocol starts from a state of U, then, no matter what adversary of $Advs$ is used to resolve the nondeterminism, some state of U' is reached with probability at least p. A progress statement is a probabilistic generalization of the *leads-to* operator of UNITY [CM88].

Example 5.1.2 It is possible to show (cf. Section 6.6) that the toy resource allocation protocol satisfies $\mathcal{R} \xrightarrow[1/2]{} _{Advs} \mathcal{M}_1$, where $\mathcal{R}$ is the set of reachable states of $M_1 \| M_2$, $\mathcal{M}_1$ is the set of states of $M_1 \| M_2$ where M_1 holds a resource, and $Advs$ is the set of fair oblivious and adversaries for $M_1 \| M_2$, i.e., the set of adversaries that are fair to each process and that do not base their choices on the outcomes of the coin flips (cf. Example 5.6.2 for a formal definition of a fair oblivious adversary). ■

Progress statements are important because, under some general conditions, they can be combined together to obtain more complex progress statements, thus allowing the decomposition of a complex problem into simpler problems.

Example 5.1.3 Suppose that in some system M whenever a request is pending (M is in a state of some set $\mathcal{P}$, a token is given eventually with probability at least $1/2$ (reaching a state of some set $\mathcal{T}$), and suppose that whenever a token is given a response is given eventually with probability at least $1/3$ (reaching a state of some set $\mathcal{G}$). That is,

$$\mathcal{P} \xrightarrow[1/2]{}_{Advs} \mathcal{T} \text{ and } \mathcal{T} \xrightarrow[1/3]{}_{Advs} \mathcal{G}. \tag{5.2}$$

Then, it is reasonable to conclude that whenever a request is pending a response is given eventually with probability at least $1/6$, i.e.,

$$\mathcal{P} \xrightarrow[1/2]{}_{Advs} \mathcal{G}. \tag{5.3}$$

This result is a consequence of the *concatenation theorem* (cf. Theorem 5.5.2). ■

Example 5.1.4 Consider the toy resource allocation protocol again. We know from Example 5.1.2 that

$$\mathcal{R} \xrightarrow[1/2]{}_{Advs} \mathcal{M}_1. \tag{5.4}$$

It is also possible to show that

$$\mathcal{R} \Rightarrow \mathcal{R} \; \mathit{Unless} \; \mathcal{M}_1, \tag{5.5}$$

where $\mathcal{R} \Rightarrow \mathcal{R} \; \mathit{Unless} \; \mathcal{M}_1$ is a UNITY [CM88] expression stating that whenever a system is in a state of $\mathcal{R}$ the system remains in a state of $\mathcal{R}$ unless a state of $\mathcal{M}_1$ is reached. This means that (5.4) is applicable from any point in the evolution of the toy resource allocation protocol, and this fact, together with the condition that every adversary is fair, is succicient to guarrantee that

$$\mathcal{R} \xrightarrow[1]{}_{Advs} \mathcal{M}_1 \tag{5.6}$$

(cf. Proposition 5.5.6). The reader familiar with UNITY may note that the combination of (5.4) and (5.5) is a probabilistic generalization of the *ensures* operator of Chandy and Misra [CM88]. ■

To see more significative applications of progress statements the reader is referred to Chapter 6, where we prove the correctness of the randomized Dining Philosophers algorithm of Lehmann and Rabin [LR81], and we prove the correctness of the randomized algorithm of Ben-Or for agreement in asynchronous networks in the presence of stopping faults [BO83]. Instead, the final part of this chapter concentrates on standard methods to specify event schemas and adversary schemas, and on the relationship between deterministic and general (randomized) adversaries. The main lesson that we learn is that for a large class of probabilistic statements it is possible to prove their validity by considering only deterministic adversaries, i.e., adversaries that do not use randomization in their choices. The reader who is reading only the first section of each chapter should move to Chapter 6 at this point and skip the rest of this section.

We said already that an event schema is a rule to associate an event with each probabilistic execution fragment. More formally, an event schema is a function that given a probabilistic execution fragment H returns an event of $\mathcal{F}_H$. However, we have not given any method to

specify an event schema. Our definition of an event schema is very general since it allows for any kind of rule to be used in determining the event associated with a probabilistic execution fragment. On the other hand, there is a specific rule which is used in most of the existing literature on randomized algorithms. Namely, given a probabilistic automaton M, a set Θ of execution fragments of M is fixed, and then, given a probabilistic execution fragment H of M, the event associated with H is $\Theta \cap \Omega_H$. We call such an event schema an *execution-based* event schema. Since the start state of a probabilistic execution fragment contains part of the history of M, and since in general we are interested in what happens only after the probabilistic execution fragment starts, we refine the definition of an execution-based event schema by associating a probabilistic execution fragment H with the event $\Theta \cap (\Omega_H \triangleright q_0^H)$, where q_0^H is the start state of H. In this way a progress statement can be stated in terms of execution-based event schemas, where Θ is the set of execution fragments of M that contain at least one occurrence of a state from U'.

To specify an adversary schema there are two main restrictions that are usually imposed. One possibility is to restrict the kind of choices that an adversary can make, and the other possibility is to restrict the on-line information that an adversary can use in making its choices. The first kind of restriction is usually achieved by fixing a set Θ of execution fragments beforehand and requiring that all the probabilistic execution fragments H generated by an adversary satisfy $\Omega_H \subseteq \Theta$. We call the corresponding adversary schema an *execution-based* adversary schema. The second kind of restriction is achieved by imposing a correlation on the choices of an adversary on different inputs. We call the corresponding adversary schema an adversaries schema with *partial on-line information*.

Example 5.1.5 An example of an execution-based adversary schema is the set of fair adversaries for n processes running in parallel. In this case Θ is the set of execution fragments of the composite system where each process performs infinitely many transitions. An example of an adversary schema with partial on-line information is the set of oblivious adversaries for the toy resource allocation protocol. Execution-based adversary schemas and adversary schemas with partial on-line information can be combined together. An example of an execution-based adversary schema with partial on-line information is the set of fair and oblivious adversaries for the toy resource protocol (cf. Example 5.6.2). ■

Exacution-based adversaries and event schemas give us a good basis to study the relationship between deterministic and general adversaries. Roughly speaking, and adversary is deterministic if it does not use randomness in its choices. Then the question is the following: "does randomness add power to an adversary?" The answer in general is "yes"; however, there are several situations of practical relevance where randomness does not add any power to an adversary. In particular, we show that randomization does not add any power when dealing with finitely satisfiable execution-based event schemas in two scenarios: execution-based adversary schemas and adversary schemas with partial on-line information.

5.2 Adversaries and Adversary Schemas

An *adversary*, also called a *scheduler*, for a probabilistic automaton M is a function $\mathcal{A}$ that takes a finite execution fragment α of M and returns a combined transition of M that leaves

from $lstate(\alpha)$. Formally,

$$\mathcal{A} : frag^*(M) \rightarrow Probs(ctrans(M))$$

such that if $\mathcal{A}(\alpha) = (s, \mathcal{P})$, then $s = lstate(\alpha)$.

An adversary is *deterministic* if it returns either transitions of M or pairs of the form $(s, \mathcal{D}(\delta))$, i.e., the next transition is chosen deterministically. Denote the set of adversaries and deterministic adversaries for a probabilistic automaton M by $Advs(M)$ and $DAdvs(M)$, respectively. We introduce deterministic adversaries explicitly because most of the existing randomized algorithms are analized against deterministic adversaries. In Section 5.7 we study the connections between deterministic adversaries and general adversaries.

As we have noted already, the correctness of an algorithm may be based on some specific assumptions on the scheduling policy that is used. Thus, in general, we are interested only in some of the adversaries of $Advs(M)$. We call a subset of $Advs(M)$ an *adversary schema*, and we use $Advs$ to denote a generic adversary schema. Section 5.6 describes in more detail possible ways to specify an adversary schema.

5.2.1 Application of an Adversary to a Finite Execution Fragment

The interaction of an adversary $\mathcal{A}$ with a probabilistic automaton M leads to a probabilistic execution fragment, where the transition enabled from each state is the transition chosen by $\mathcal{A}$. Given a finite execution fragment α of M, the probabilistic execution of M under $\mathcal{A}$ with starting condition α, denoted by $prexec(M, \mathcal{A}, \alpha)$, is the unique probabilistic execution fragment H of M such that

1. $start(H) = \{\alpha\}$, and
2. for each state q of H, the transition tr_q^H is $q \frown \mathcal{A}(q)$.

Condition 2 ensures that the transition enabled from every state q of H is the transition chosen by $\mathcal{A}$. It is a simple inductive argument to show that H is well defined.

5.2.2 Application of an Adversary to a Finite Probabilistic Execution Fragment

From the theoretical point of view, we can generalize the idea of the interaction between an adversary and a probabilistic automaton by assuming that the start condition is a finite probabilistic execution fragment of M. In this case the adversary works from all the points of extension of the starting condition. The resulting probabilistic execution fragment should be an extension of the starting condition. Formally, if H is a finite probabilistic execution fragment of M, then the probabilistic execution of M under $\mathcal{A}$ with starting condition H, denoted by $prexec(M, \mathcal{A}, H)$, is the unique probabilistic execution fragment H' of M such that

1. $start(H') = start(H)$, and
2. for each state q of H', if q is a state of H, then $tr_q^{H'}$ is

$$p\left(tr_q^H \upharpoonright acts(H)\right) + (1-p)\left(q \frown \mathcal{A}(q)\right),$$

Figure 5-2: An example of the action of an adversary on a probabilistic execution fragment.

where

$$p = \frac{P_H[C_q]}{P_{H'}[C_q]} P_q^H[acts(H)],$$

and if q is not a state of H, then $tr_q^{H'}$ is $q \frown \mathcal{A}(q)$.

Once again, it is a simple inductive argument to show that H' is well defined.

Example 5.2.1 (Extension of a finite probabilistic execution fragment) Before proving that H' is an extension of H, we describe in more detail how the definition above works. The difficult case is for those states q of H' that are also states of H. Consider the example of Figure 5-2. Let $\mathcal{A}$ choose $q_0 \xrightarrow{a} q$ on input q_0, choose $q \xrightarrow{b} q_2$ on input q, and choose δ on all other inputs. The probabilistic execution fragment H' of Figure 5-2 is the result of the action of $\mathcal{A}$ on the probabilistic execution fragment H of Figure 5-2. In H' there are two ways to reach q: one way is by means of transitions of H, and the other way is by means of transitions due to $\mathcal{A}$ that originate from q_0. Thus, a fraction of the probability of reaching q in H' is due to H, while another fraction is due to the effect of $\mathcal{A}$ on H. The weight with which the transition tr_q^H is considered in H' is the first fraction of the probability of reaching q, which is expressed by $P_H[C_q]/P_{H'}[C_q]$. In our example the fraction is $1/2$. However, in our example the transition tr_q^H may also leads to δ with probability $1/2$, and the part of tr_q^H that leads to δ should be handled by $\mathcal{A}$. For this reason in the left term of the definition of $tr_q^{H'}$ we discard δ from tr_q^H and we add a multiplicative factor $P_q^H[acts(H)]$ to the weight. Thus, in our example, three quarters of the transition leaving from q in H' are controlled by $\mathcal{A}$. Note that the probability of reaching q_1 from q_0 is the same in H and H'. ■

Proposition 5.2.1 *Let M be a probabilistic automaton, and let $\mathcal{A}$ be an adversary for M. Then, for each finite probabilistic execution fragment H of M, the probabilistic execution fragment generated by $\mathcal{A}$ from H is an extension of H, i.e.,*

$$H \leq prexec(M, \mathcal{A}, H).$$

Proof. Denote $prexec(M, \mathcal{A}, H)$ by H'. We need to prove that for each state q of H,

$$P_H[C_q] \leq P_{H'}[C_q]. \tag{5.7}$$

If q is the start state of H, then q is also the start state of H', and (5.7) is satisfied trivially.

Consider now a state qas of H that is not the start state of H. Then q is a state of H. From the definition of the probability of a cone,

$$P_{H'}[C_{qas}] = P_{H'}[C_q] P_q^{H'}[(a, qas)]. \tag{5.8}$$

From the definition of $tr_q^{H'}$,

$$P_q^{H'}[(a,qas)] = \frac{P_H[C_q]}{P_{H'}[C_q]} P_q^H[(a,qas)] + \left(1 - \frac{P_H[C_q]}{P_{H'}[C_q]} P_q^H[acts(H)]\right) P_{\mathcal{A}(q)}[(a,qas)]. \quad (5.9)$$

Here we have also simplified the expression $P_q^H[acts(H)]$ in the first term as we did in the proof of Proposition 4.3.5 (Expressions (4.33) and (4.34)). We will not mention this simplification any more in the thesis.

If we remove the second term from the right expression of Equation (5.9), turning Equation (5.9) into an inequality, we obtain

$$P_q^{H'}[(a,qas)] \geq \frac{P_H[C_q]}{P_{H'}[C_q]} P_q^H[(a,qas)]. \quad (5.10)$$

By using (5.10) in (5.8), and simplifying the factor $P_{H'}[C_q]$, we obtain

$$P_{H'}[C_{qas}] \geq P_H[C_q] P_q^H[(a,qas)]. \quad (5.11)$$

The right part of (5.11) is $P_H[C_{qas}]$. Thus, we conclude

$$P_{H'}[C_{qas}] \geq P_H[C_{qas}]. \quad (5.12)$$

■

5.3 Event Schemas

In the informal description of a probabilistic statement we said that we need a rule to associate an event with each probabilistic execution fragment. This is the purpose of an event schema. An *event schema* for a probabilistic automaton M, denoted by e, is a function that associates an event of $\mathcal{F}_H$ with each probabilistic execution fragment H of M. An event schema e is *finitely satisfiable* iff for each probabilistic execution fragment H the event $e(H)$ is finitely satisfiable. Union, intersection and complementation of event schemas are defined pointwise. Similarly, conditional event schemas are defined pointwise.

The best way to think of an event schema is just as a rule to associate an event with each probabilistic execution fragment. Although in most of the practical cases the rule can be specified by a set of executions (cf. Section 5.3.2), part of our results do not depend on the actual rule, and thus they would hold even if for some reason in the future we need to study different rules. Moreover, event schemas allow us to simplify the notation all over.

5.3.1 Concatenation of Event Schemas

If e is a finitely satisfiable event schema, i.e., for each probabilistic execution fragment H the event $e(H)$ can be expressed as a union of cones, then it means that in every execution of $e(H)$ it is possible to identify a finite point where the property denoted by e is satisfied. Sometimes we may be interested in checking whether a different property, expressed by another event schema, is satisfied eventually once the property expressed by e is satisfied. That is, we want to concatenate two event schemas.

Formally, let e_1, e_2 be two event schemas for a probabilistic automaton M where e_1 is finitely satisfiable, and let *Cones* be a function that associates a set $Cones(H)$ with each probabilistic execution fragment H of M such that $Cones(H)$ is a characterization of $e_1(H)$ as a union of disjoint cones, i.e., $e_1(H) = \cup_{q \in Cones(H)} C_q$, and for each $q_1, q_2 \in Cones(H)$, if $q_1 \neq q_2$, then $C_{q_1} \cap C_{q_2} = \emptyset$. Informally, $Cones(H)$ identifies the points where the event denoted by $e_1(H)$ is satisfied, also called *points of satisfaction*.

The *concatenation* $e_1 \circ_{Cones} e_2$ of e_1 and e_2 via *Cones* is the function e such that, for each probabilistic execution fragment H of M,

$$e(H) \stackrel{\triangle}{=} \bigcup_{q \in Cones(H)} e_2(H|q). \tag{5.13}$$

Proposition 5.3.1 *The concatenation of two event schemas is an event schema. That is, if $e = e_1 \circ_{Cones} e_2$, then e is an event schema.*

Proof. Consider a probabilistic execution fragment H. From Proposition 4.2.11 each set $e_2(H|q)$ is an event of $\mathcal{F}_H$. From the closure of a σ-field under countable union, $e(H)$ is an event of $\mathcal{F}_H$. ■

Proposition 5.3.2 $P_H[e_1 \circ_{Cones} e_2(H)] = \sum_{q \in Cones(H)} P_H[C_q] P_{H|q}[e_2(H|q)]$.

Proof. Since $Cones(H)$ represents a collection of disjoint cones, from (5.13) we obtain

$$P_H[e_1 \circ_{Cones} e_2(H)] = \sum_{q \in Cones(H)} P_H[e_2(H|q)]. \tag{5.14}$$

From Proposition 4.2.11, for each $q \in Cones(H)$

$$P_H[e_2(H|q)] = P_H[C_q] P_{H|q}[e_2(H|q)]. \tag{5.15}$$

By substituting (5.15) in (5.14) we obtain the desired result. ■

5.3.2 Execution-Based Event Schemas

Our definition of an event schema is very general; on the other hand, most of the existing work on randomized algorithms is based on a very simple rule to associate an event with each probabilistic execution. Namely, a set Θ of execution fragments of M is chosen beforehand, and then, given a probabilistic execution fragment H, the event associated with H is the $\Theta \frown \Omega_H$. We call this class of event schemas *execution-based*. We have chosen to give a more general definition of an event schema for two main reasons:

1. The concatenation Theorem of Section 5.4.1 (Theorem 5.4.2) does not rely on the fact that an event schema is execution-based, but rather on the fact that it is finitely satisfiable. Thus, if in the future some different kinds of event schemas will become relevant, here we have already the machinery to deal with them.

2. The event schemas that we use later to define a progress statement (cf. Section 5.5) are not execution-based according to the informal description given above. Specifically, the start state of a probabilistic execution fragment of M is a finite execution fragment of

M, i.e., it contains some history of M, and such history is not considered in determining whether there is some progress. On the other hand, it is plausible that sometimes we want to consider also the history encoded in the start state of a probabilistic execution fragment. Thus, the more general definition of an event schema still helps.

Nevertheless, execution-based adversary schemas are easier to understand and enjoy properties that do not hold for general adversary schemas (cf. Section 5.7). For this reason we give a formal definition of an execution-based adversary schema, where we also assume that the history encoded in the start state of a probabilistic execution fragment is eliminated.

Let Θ be a set of extended execution fragments of M. An event schema e for a probabilistic automaton M is Θ-*based* iff for each probabilistic execution fragment H of M, $e(H) = \Theta \cap (\Omega_H \triangleright q_0^H)$. An event schema e for a probabilistic automaton M is *execution-based* iff there exists a set Θ of extended execution fragments of M such that e is Θ-based.

5.4 Probabilistic Statements

Given a probabilistic automaton M, an event schema e, an adversary $\mathcal{A}$, and a finite execution fragment α, it is possible to compute the probability $P_{prexec(M,\mathcal{A},\alpha)}[e(prexec(M,\mathcal{A},\alpha))]$ of the event denoted by e when M starts from α and interacts with $\mathcal{A}$. As a notational convention, we abbreviate the expression above by $P_{M,\mathcal{A},\alpha}[e]$. Moreover, when M is clear from the context we write $P_{\mathcal{A},\alpha}[e]$, and we write $P_{\mathcal{A}}[e]$ if M has a unique start state and α is chosen to be the start state of M.

We now have all the machincery necessary to define a probabilistic statement. A *probabilistic statement* for a probabilistic automaton M is an expression of the form $\Pr_{Advs,\Theta}(e)\ \mathcal{R}\ p$, where $Advs$ is an adversary schema of M, Θ is a set of starting conditions, i.e., a set of finite execution fragments of M, e is an event schema for M, and $\mathcal{R}$ is a relation among $=$, $\leq$, and $\geq$. A probabilistic statement $\Pr_{Advs,\Theta}(e)\ \mathcal{R}\ p$ is valid for M iff for each adversary $\mathcal{A}$ of $Advs$ and each starting condition α of Θ, $P_{\mathcal{A},\alpha}[e]\ \mathcal{R}\ p$, i.e.,

$$\Pr_{Advs,\Theta}(e)\ \mathcal{R}\ p \text{ iff } \forall_{\mathcal{A}\in Advs}\forall_{\alpha\in\Theta} P_{\mathcal{A},\alpha}[e]\ \mathcal{R}\ p. \tag{5.16}$$

Proposition 5.4.1 *Some trivial properties of probabilistic statements are the following.*

1. *If* $p_1\ \mathcal{R}\ p_2$ *then* $\Pr_{Advs,\Theta}(e)\ \mathcal{R}\ p_1$ *implies* $\Pr_{Advs,\Theta}(e)\ \mathcal{R}\ p_2$.
2. *If* $Advs_1 \subseteq Advs_2$ *and* $\Theta_1 \subseteq \Theta_2$, *then* $\Pr_{Advs_1,\Theta_1}(e)\ \mathcal{R}\ p$ *implies* $\Pr_{Advs_2,\Theta_2}(e)\ \mathcal{R}\ p$. ■

5.4.1 The Concatenation Theorem

We now study an important property of probabilistic statements applied to the concatenation of event schemas. Informally, we would like to derive properties of the concatenation of two event schemas from properties of the event schemas themselves. The idea that we want to capture is expressed by the sentence below and is formalized in Theorem 5.4.2.

> "*If* e_1 *is satisfied with probability at least* p_1, *and from every point of satisfaction of* e_1, e_2 *is satisfied with probability at least* p_2, *then the concatenation of* e_1 *and* e_2 *is satisfied with probability at least* $p_1 p_2$."

Theorem 5.4.2 *Consider a probabilistic automaton M. Let*

1. $\Pr_{Advs,\Theta}(e_1)\ \mathcal{R}\ p_1$ *and,*
2. *for each* $\mathcal{A} \in Advs$, $q \in \Theta$, *let* $\Pr_{Advs,Cones(prexec(M,\mathcal{A},q))}(e_2)\ \mathcal{R}\ p_2$.

Then, $\Pr_{Advs,\Theta}(e_1 \circ_{Cones} e_2)\ \mathcal{R}\ p_1 p_2$.

Proof. Consider an adversary $\mathcal{A} \in Advs$ and any finite execution fragment $q \in \Theta$. Let $H = prexec(M, \mathcal{A}, q)$. From Proposition 5.3.2,

$$P_H[e_1 \circ_{Cones} e_2(H)] = \sum_{q' \in Cones(H)} P_H[C_{q'}] P_{H|q'}[e_2(H|q')]. \tag{5.17}$$

Consider an element q' of $Cones(H)$. It is a simple inductive argument to show that

$$H|q' = prexec(M, \mathcal{A}, q'). \tag{5.18}$$

Thus, from our second hypothesis,

$$P_{H|q'}[e_2(H|q')]\ \mathcal{R}\ p_2. \tag{5.19}$$

By substituting (5.19) in (5.17), we obtain

$$P_H[e_1 \circ_{Cones} e_2(H)]\ \mathcal{R}\ p_2 \sum_{q' \in Cones(e_1(H))} P_H[C_{q'}]. \tag{5.20}$$

By using the fact that $Cones(H)$ is a characterization of $e_1(H)$ as a disjoint union of cones, Equation (5.20) can be rewritten into

$$P_H[e_1 \circ_{Cones} e_2(H)]\ \mathcal{R}\ p_2 P_H[e_1(H)]. \tag{5.21}$$

From the first hypothesis, $P_H[e_1(H)]\ \mathcal{R}\ p_1$; therefore, from Proposition 5.4.1,

$$P_H[e_1 \circ_{Cones} e_2(H)]\ \mathcal{R}\ p_1 p_2. \tag{5.22}$$

This completes the proof. ■

5.5 Progress Statements

In this section we give examples of probabilistic statements, which we call progress statements, that play an important role in the analysis of algorithms. Progress statements are formalizations of statements that are used generally for the informal analysis of randomized algorithms; however, many other statements can be defined depending on specific applications. We show also how to derive complex statements by concatenating several simple statements.

5.5.1 Progress Statements with States

Let U and U' be sets of states of a probabilistic automaton M. A common informal statement is the following.

> "*Whenever the system is in a state of U, then, under any adversary $\mathcal{A}$ of $Advs$, the probability that a state of U' is reached is at least p.*"

The probability p is usually 1. In this thesis we consider the more general statement where p is required only to be greater than 0. We represent the statement concisely by writing

$$U \xrightarrow[p]{}_{Advs} U', \tag{5.23}$$

where $Advs$ is an adversary schema. We call (5.23) a *progress statement* since, if we view U' as a better condition than U, then (5.23) states that from U it is possible to have some progress with probability at least p. The reader familiar with UNITY [CM88] may note that a progress statement is a probabilistic generalization of the *leads-to* operator of UNITY.

Let us concentrate on the formal meaning of (5.23). Let $e_{U'}$ be an event schema that given a probabilistic execution fragment H returns the set of extended executions α of Ω_H such that a state of U' is reached in $\alpha \triangleright q_0^H$ (recall that q_0^H is the start state of H). Then (5.23) is the probabilistic statement

$$\Pr_{Advs,U}(e_{U'}) \geq p. \tag{5.24}$$

Note that the starting conditions of statement (5.24) are just states of M, i.e., they do not contain any past history of M except for the current state. This is because when we reason informally about algorithms we do not talk usually about the past history of a system. However, if we want to concatenate two progress statements according to Theorem 5.4.2, then we need to consider the past history explicitly, and thus a better probabilistic statement for (5.23) would be

$$\Pr_{Advs,\Theta_U}(e_{U'}) \geq p, \tag{5.25}$$

where Θ_U is the set of finite execution fragments of M whose last state is a state of U. So, why can we, and indeed do people, avoid to deal with the past history explicitly? The point is that (5.24) and (5.25) are equivalent for most of the adversary schemas that are normally used.

5.5.2 Finite History Insensitivity

An adversary schema $Advs$ for a probabilistic automaton M is *finite-history-insensitive* iff for each adversary $\mathcal{A}$ of $Advs$ and each finite execution fragment α of M, there exists an adversary $\mathcal{A}'$ of $Advs$ such that for each execution fragment α' of M with $fstate(\alpha') = lstate(\alpha)$, $\mathcal{A}'(\alpha') = \mathcal{A}(\alpha \frown \alpha')$. In other words, $\mathcal{A}'$ does even though $\mathcal{A}'$ does not know the finite history α.

Lemma 5.5.1 *Let $Advs$ be a finite-history-insensitive adversary schema for a probabilistic automaton M. Then (5.24) and (5.25) are equivalent probabilistic statements.*

Proof. From Proposition 5.4.1, since $U \subseteq \Theta_U$, Statement (5.25) implies Statement (5.24) trivially. Conversely, suppose that Statement (5.24) is valid. Consider an adversary $\mathcal{A}$ of $Advs$, and consider an element q of Θ_U. Let $\mathcal{A}_q$ be an adversary of $Advs$ such that for each execution fragment q' of M with $fstate(q') = lstate(q)$, $\mathcal{A}_q(q') = \mathcal{A}(q \frown q')$. We know that $\mathcal{A}_q$ exists since $Advs$ is finite-history-insensitive. It is a simple inductive argument to show that

$$prexec(M, \mathcal{A}_q, lstate(q)) = prexec(M, \mathcal{A}, q) \triangleright q. \tag{5.26}$$

Moreover,

$$P_{prexec(M,\mathcal{A},q)}[C_q] = 1. \tag{5.27}$$

From the definition of $e_{U'}$, since the start state of $prexec(M, \mathcal{A}, q)$ is q,

$$e_{U'}(prexec(M, \mathcal{A}_q, lstate(q))) = e_{U'}(prexec(M, \mathcal{A}, q)) \triangleright q. \tag{5.28}$$

Thus, from Proposition 4.2.12 and (5.27),

$$P_{\mathcal{A},q}[e_{U'}] = P_{\mathcal{A}_q, lstate(q)}[e_{U'}]. \tag{5.29}$$

From hypothesis,

$$P_{\mathcal{A}_q, lstate(q)}[e_{U'}] \geq p, \tag{5.30}$$

and thus, from (5.29), $P_{\mathcal{A},q}[e_{U'}] \geq p$. This shows that Statement (5.25) is valid. ■

5.5.3 The Concatenation Theorem

We now start to compose (simple) progress statements to derive other (more complex) progress statements. This allows us to decompose a complex problems into simpler problems that can be solved separately. The examples of Chapter 6 contain explicit use of the concatenation theorem of this section.

Suppose that from U we can reach U' with probability at least p, and that from U' we can reach U'' with probability at least p'. Then, it is reasonable that from U we can reach U'' with probability at least pp'. This result is an instantiation of the concatenation theorem of Section 5.4.1.

Theorem 5.5.2 *Let $Advs$ be a finite-history-insensitive adversary schema. Then,*

$$U \xrightarrow[p]{}_{Advs} U' \text{ and } U' \xrightarrow[p']{}_{Advs} U'' \text{ imply } U \xrightarrow[pp']{}_{Advs} U''.$$

Proof. Consider the event schemas $e_{U'}$ and $e_{U''}$. Let $Cones$ be the function that associates with each probabilistic execution fragment H the set

$$Cones(H) \triangleq \{q \mid lstate(q \triangleright q_0) \in U', \nexists_{q' < (q \triangleright q_0)}\, lstate(q') \in U'\}. \tag{5.31}$$

It is easy to check that $Cones(H)$ is a characterization of $e_{U'}$ as a disjoint union of cones. Then, directly from the definitions, for each execution fragment H,

$$e_{U'} \circ_{Cones} e_{U''}(H) \subseteq e_{U''}(H). \tag{5.32}$$

Informally, the left expression represents the property of reaching a state of U'' passing through a state of U', while the right expression represents the property of reaching a state of U'' without passing necessarily through a state of U'.

From Lemma 5.5.1, for each probabilistic execution fragment H, each adversary $\mathcal{A}$ of $Advs$, and each element q of $Cones(H)$, since $lstate(q) \in U'$,

$$P_{\mathcal{A},q}[e_{U''}] \geq p'. \tag{5.33}$$

From hypothesis, (5.33), and Theorem 5.4.2 (concatenation of two event schemas),

$$\Pr_{Advs,U}(e_{U'} \circ_{Cones} e_{U''}) \geq pp'. \tag{5.34}$$

From (5.32) and (5.34),

$$\Pr_{Advs,U}(e_{U''}) \geq pp'. \tag{5.35}$$

This shows that $U \underset{pp'}{\longrightarrow}_{Advs} U''$. ■

Proposition 5.5.3 *Other trivial properties of progress statements are the following.*

1. $U \underset{1}{\longrightarrow} U$.
2. *If* $U_1 \underset{p_1}{\longrightarrow} U_1'$ *and* $U_2 \underset{p_2}{\longrightarrow} U_2'$, *then* $U_1 \cup U_2 \underset{min(p_1,p_2)}{\longrightarrow} U_1' \cup U_2'$. ■

5.5.4 Progress Statements with Actions

Progress statements can be formulated also in terms of actions rather than states. Thus, if V is a set of actions, we could write

$$U \underset{p}{\longrightarrow}_{Advs} V \tag{5.36}$$

meaning that starting from any state of U and under any adversary of $Advs$, with probability at least p an action from V occurs. Formally, let e_V be an event schema that given a probabilistic execution fragment H returns the set of executions α of Ω_H such that an action from V occurs in $\alpha \triangleright q_0^H$. Then (5.36) is the probabilistic statement

$$\Pr_{Advs,U}(e_V) \geq p. \tag{5.37}$$

Similarly, we can change the left side of a progress statement. Thus, we can write

$$V \underset{p}{\longrightarrow}_{Advs} U \tag{5.38}$$

meaning that starting from any point where an action from V occurred and no state of U is reached after the last occurrence of an action from V, a state of U is reached with probability at least p. In other words, after an action from V occurs, no matter what the system has done, a state of U is reached with probability at least p. Formally, let $\Theta_{V,U}$ be the set of finite execution fragments of M where an action from V occurs and no state of U occurs after the last occurrence of an action from V. Then (5.38) is the probabilistic statement

$$\Pr_{Advs,\Theta_{V,U}}(e_U) \geq p. \tag{5.39}$$

Finally, we can consider statements involving only sets of actions. Thus, the meaning of $V \xrightarrow[p]{} _{Advs} V'$ would be the probabilistic statement

$$\Pr_{Advs,\Theta_{V,V'}}(e_V) \geq p, \tag{5.40}$$

where $\Theta_{V,V'}$ is the set of finite execution fragments of M where an action from V occurs and no action from V' occurs after the last occurrence of an action from V.

The concatenation theorem extendeds easily to the new kinds of progress statements.

Theorem 5.5.4 *Let Advs be a finite-history-insensitive adversary schema, and let X, X' and X'' be three sets, each one consisting either of actions of M only or states of M only. Then,*

$$X \xrightarrow[p_1]{} _{Advs} X' \text{ and } X' \xrightarrow[p_2]{} _{Advs} X'' \text{ imply } X \xrightarrow[p_1 p_2]{} _{Advs} X''.$$

Proof. This proof is similar to the proof of Theorem 5.5.2, and thus it is left to the reader. Observe that finite-history-insensitivity is not necessary if X' is a set of actions. ■

5.5.5 Progress Statements with Probability 1

Usually we are interested in progress properties that hold with probability 1. A useful result is that in most cases progress with probability 1 can be derived from progress with any probability p such that $0 < p < 1$. Specifically, under the condition that an adversary never chooses δ when the left side of a given progress statement is satisfied and the right side of the same progress statement is not satisfied,

1. if the left element of the progress statement is a set of actions, then progress is achieved with probability 1;

2. if the left element of the progress statement is a set of states U, the adversary schema is finite-history-insensitive, and the system remains in a state of U unless the right side of the statement is satisfied, then progress is achieved with probability 1.

Proposition 5.5.5 *Suppose that $V \xrightarrow[p]{} _{Advs} X$, and suppose that $\delta \notin \Omega_{\mathcal{A}(q)}$ for each adversary $\mathcal{A}$ of Advs and each element q of $\Theta_{V,X}$. Then $V \xrightarrow[1]{} _{Advs} X$.*

Proof. We give the proof for the case where X is a set of states. The other proof is similar. Denote X by U.

Consider an element q_0 of $\Theta_{V,U}$ and an adversary $\mathcal{A}$ of $Advs$. Let H be $prexec(M, \mathcal{A}, q_0)$, and let $p' = P_H[e_U(H)]$. We know from hypothesis that $p' \geq p$. Suppose by contradiction that $p' < 1$. Let Θ be the set of finite execution fragments q of M such that $q_0 \leq q$, $lstate(q) \in U$, and no state of U occurs in any proper prefix of $q \triangleright q_0$. Then Θ is a characterization of $e_U(H)$ as a union of disjoint cones. Thus,

$$P_H[e_U(H)] = \sum_{q \in \Theta} P_H[C_q]. \tag{5.41}$$

Let ϵ be any real number such that $0 \leq \epsilon \leq p'$. Then, from (5.41) and the definition of p', it is possible to find a natural number k_ϵ such that.

$$\sum_{q \in \Theta || q| \leq k_\epsilon} P_H[C_q] \geq (p' - \epsilon). \tag{5.42}$$

Let Θ_ϵ be the set of states q of H such that $|q| = k_\epsilon$ and no prefix of q is in Θ. That is, Θ_ϵ is the set of states of H of length k_ϵ that are not within any cone C_q of $e_U(H)$ where $|q| \leq k_\epsilon$. Equation (5.41) can be rewritten as

$$P_H[e_U(H)] = \left(\sum_{q \in \Theta || q| \leq k_\epsilon} P_H[C_q] \right) + \left(\sum_{q \in \Theta_\epsilon} P_H[C_q] P_H[e_U(H)|C_q] \right). \tag{5.43}$$

Observe that for each state q of Θ_ϵ, since a state of U' is not reached yet, q is an element of $\Theta_{V,U}$. Moreover, $prexec(M, \mathcal{A}, q) = H|q$ (simple inductive argument). Thus, from Proposition 4.2.11 and hypothesis, $P_H[e_U(H)|C_q] \geq p$, and (5.43) can be rewritten into

$$P_H[e_U(H)] \geq \left(\sum_{q \in \Theta || q| \leq k_\epsilon} P_H[C_q] \right) + \left(\sum_{q \in \Theta_\epsilon} P_H[C_q] p \right). \tag{5.44}$$

Observe that $\sum_{q \in \Theta || q| \leq k_\epsilon} P_H[C_q] + \sum_{q \in \Theta_\epsilon} P_H[C_q] = 1$. This follows from the fact that if a state q of H does not have any prefix in Θ, then $q \in \Theta_{V,X}$, which in turn means that $\delta \notin \Omega_q^H$. In other words, in H it is not possible to stop before reaching either a state of $\{q \in \Theta \mid |q| \leq k_\epsilon\}$ or a state of Θ_ϵ. Thus, by using (5.42) in (5.44) we obtain

$$P_H[e_U(H)] \geq (p' - \epsilon) + (1 - (p' - \epsilon))p. \tag{5.45}$$

After simple algebraic manipulations, Equation (5.45) can be rewritten into

$$P_H[e_U(H)] \geq p' + p(1 - p') - \epsilon(1 - p). \tag{5.46}$$

If we choose ϵ such that $0 < \epsilon < p(1-p')/(1-p)$, which exists since $p' < 1$, then Equation (5.46) shows that $P_H[e_U(H)] > p'$. This contradicts the fact that $p' < 1$. Thus, $P_H[e_U(H)] = 1$. ■

For the next proposition we define the statement *U Unless X*, where U is a set of states and X is either a set of states only or a set of actions only. The statement is true for a probabilistic automaton M iff for each transition $(s, \mathcal{P})$ of M, if $s \in U - X$ then for each $(a, s') \in \Omega$ either $a \in X$, or $s' \in U \cup X$. That is, once in U, the probabilistic automaton M remains in U until the condition expressed by X is satisfied.

Proposition 5.5.6 *Suppose that* $U \xrightarrow[p]{} _{Advs} X$, *U Unless X, Advs is finite-history-insensitive, and* $\delta \notin \Omega_{\mathcal{A}}(s)$ *for each adversary* $\mathcal{A}$ *of Advs and each state s of U. Then,* $U \xrightarrow[1]{} _{Advs} X$.

Proof. This proof is similar to the proof of Proposition 5.5.5. The main difference is that the passage from Equation (5.43) to Equation (5.44) is justified by using finite-history-insensitivity as in the proof of Proposition 5.5.1. ■

5.6 Adversaries with Restricted Power

In Section 5.2 we have defined adversary schemas to reduce the power of an adversary; however, we have not described any method to specify how the power of an adversary is reduced. In this section we show two methods to reduce the power of an adversary. The first method, which is the most commonly used, reduces the kind of choices that an adversary can make; the second method, which is used in informal arguments but is rarely formalized, reduces the on-line information used by an adversary to make a choice. The two specification methods are used in Section 5.7 to study the relationship between deterministic and randomized adversaries.

5.6.1 Execution-Based Adversary Schemas

If n processes run in parallel, then a common requirement of a scheduler is to be fair to all the processes. This means that whenever an adversary resolves the nondeterminism and leads to a probabilistic execution fragment H, in all the executions of Ω_H each one of the n processes performs infinitely many transitions. More generally, a set Θ of extended execution fragments of M is set beforehand, and then an adversary is required to lead only to probabilistic execution fragments whose corresponding sample space is a subset of Θ.

Formally, let Θ be a set of extended execution fragments of M. Let $Advs_\Theta$ be the set of adversaries $\mathcal{A}$ such that for each finite execution fragment q of M, $\Omega_{prexec(M,\mathcal{A},q)} \subseteq \Theta$. Then $Advs_\Theta$ is called Θ-based. An adversary schema $Advs$ is *execution-based* iff there exists a set Θ of extended execution fragments of M such that $Advs$ is Θ-based.

The notion of finite-history-insensitivity can be reformulated easily for execution-based adversary schemas. Define Θ to be finite-history-insensitive iff for each extended execution fragment α of M and each finite execution fragment α' of M such that $lstate(\alpha') = fstate(\alpha)$, if $\alpha' \frown \alpha \in \Theta$ then $\alpha \in \Theta$. It is easy to verify that if Θ is finite-history-insensitive, then $Advs_\Theta$ is finite-history-insensitive.

5.6.2 Adversaries with Partial On-Line Information

Sometimes, like in the case of the toy resource allocation protocol, an adversary cannot base its choices on the whole history of a system if we want to guarantee progress. In other words, some part of the history is not visible to the adversary.

Example 5.6.1 (Off-line scheduler) The simplest kind of adversary for n processes that run in parallel is an adversary that fixes in advance the order in which the processes are scheduled. This is usually called an *off-line scheduler* or an *oblivious adversary*. Thus, at each point α the next transition to be scheduled depends only on the ordered sequence of processes that are scheduled in α.

To be more precise, the transition scheduled by the adversary depends also on the state that is reached by α, i.e., $lstate(\alpha)$, since a specific process may enable different transitions from different states. This means that if α_1 and α_2 are equivalent in terms of the ordered sequence of processes that are scheduled, the oblivious constraint says only that the transitions chosen by the adversary in α_1 and α_2 must be correlated, i.e., they must be transitions of the same process. ■

The formal definition of an adversary with partial on-line information for a probabilistic automaton M is given by specifying two objects:

1. an *equivalence relation* that specifies for what finite execution fragments of M the choices of an adversary must be correlated;

2. a collection of *correlation functions* that specify how the transitions chosen by an adversary must be correlated.

Let $\equiv$ be an equivalence relation between finite execution fragments of M, and let F be a family of functions parameterized over pairs of equivalent execution fragments. Each function

$f_{\alpha\alpha'}$ takes a combined transition of M leaving from $lstate(\alpha)$ and returns a combined transition of M leaving from $lstate(\alpha')$ such that

1. $f_{\alpha'\alpha}(f_{\alpha\alpha'}(tr)) = tr$;
2. $f_{\alpha\alpha'}(\sum_{i\in I} p_i tr_i) = \sum_{i\in I} p_i f_{\alpha\alpha'}(tr_i)$.

The pair $(\equiv, F)$ is called an *oblivious relation*. An adversary $\mathcal{A}$ is *oblivious relative to* $(\equiv, F)$ iff for each pair of equivalent execution fragments of M, $\alpha \equiv \alpha'$, $\mathcal{A}(\alpha') = f_{\alpha\alpha'}(\mathcal{A}(\alpha))$. An adversary schema $Advs$ is said to be with *partial on-line information* iff there exists an oblivious relation $(\equiv, F)$ such that $Advs$ is the set of adversaries for M that are oblivious relative to $(\equiv, F)$.

Condition 1 is used to guarantee that there are oblivious adversaries relative to $(\equiv, F)$; Condition 2 is more technical and is used to guarantee that there are oblivious adversaries relative to $(\equiv, F)$ that do not use randomization in their choices. Condition 2 is needed mainly to prove some of the results of Section 5.7.

Adversaries with partial on-line information and execution-based adversaries can be combined together easily. Thus, an adversary schema $Advs$ is said to be execution-based and with partial on-line information iff there exists an execution-based adversary schema $Advs'$ and a pair $(\equiv, F)$ such that $Advs$ is the set of adversaries of $Advs'$ that are oblivious relative to $(\equiv, F)$.

Example 5.6.2 (Adversaries for the toy-resource allocation protocol) The fair oblivious adversaries for the toy resource allocation protocol are an example of an execution-based adversary schema with partial on-line information. The set Θ is the set of executions of $M_1 \| M_2$ where both M_1 and M_2 perform infinitely many transitions. Two finite execution fragments α_1 and α_2 are equivalent iff the ordered sequences of the processes that perform a transition in α_1 and α_2 are the same. Let $\alpha_1 \equiv \alpha_2$, and let, for $i = 1, 2$, $tr_{i,1}$ and $tr_{i,2}$ be the transitions of M_1 and M_2, respectively, enabled from $lstate(\alpha_i)$. Then $f_{\alpha_1\alpha_2}(tr_{1,1}) = tr_{2,1}$ and $f_{\alpha_1\alpha_2}(tr_{1,2}) = tr_{2,2}$.

Another execution-based adversary schema with partial on-line information that works for the toy resource allocation protocol is obtained by weakening the equivalence relation so that an adversary cannot see only those coins that have not been used yet, i.e., those coins that have been flipped but have not been used yet to check whether the chosen resource is free. ■

5.7 Deterministic versus Randomized Adversaries

In our definition of an adversary we have allowed the use of randomness for the resolution of the nondeterminism in a probabilistic automaton M. This power that we give to an adversary corresponds to the possibility of combining transitions of M in the definition of a probabilistic execution fragment. From the formal point of view, randomized adversaries allow us to model a randomized environment and to state and prove the closure of probabilistic execution fragments under projection (Proposition 4.3.4). However, one question is still open:

Are randomized adversaries more powerful than deterministic adversaries?

That is, if an algorithm performs well under any deterministic adversary, does it perform well under any adversary as well, or are there any randomized adversaries that can degrade the performance of the algorithm? In this section we want to show that in practice randomization

does not add any power to an adversary. We say "in practice" because it is easy to build examples where randomized adversaries are more powerful than deterministic adversaries, but those examples do not seem to be relevant in practice.

Example 5.7.1 (Randomization adds power) Consider an event schema e that applied to a probabilistic execution fragment H returns Ω_H if H can be generated by a deterministic adversary, and returns $\emptyset$ otherwise. Clearly, if M is a nontrivial probabilistic automaton, the probability of e is at least 1 under any deterministic adversary, while the probability of e can be 0 under some randomized adversary; thus, randomization adds power to the adversaries. However, it is unlikely that a realistic event schema has the structure of e. Another less pathological example appears in Section 5.7.2 (cf. Example 5.7.2). ■

We consider the class of execution-based event schemas, and we restrict our attention to the subclass of finitely satisfiable, execution-based event schemas. We show that randomization does not add any power for finitely satisfiable, execution-based event schemas under two scenarios: execution-based adversary schemas, and execution-based adversary schemas with partial on-line information. In the second case we need to be careful (cf. Example 5.7.2).

Informally, a randomized adversary can be seen as a convex combination of deterministic adversaries, and thus a randomized adversary satisfies the same probability bounds of a deterministic adversary. However, there are uncountably many deterministic adversaries, and thus from the formal point of view some more careful analysis is necessary.

5.7.1 Execution-Based Adversary Schemas

Proposition 5.7.1 *Let $Advs$ be an execution-based adversary schema for M, and let $Advs_D$ be the set of deterministic adversaries of $Advs$. Let e be a finitely-satisfiable, execution-based, event schema for M. Then, for every set Θ of finite execution fragments of M, every probability p, and every relation $\mathcal{R}$ among $\leq$, $=$, $\geq$, $\Pr_{Advs,\Theta}(e)\ \mathcal{R}\ p$ iff $\Pr_{Advs_D,\Theta}(e)\ \mathcal{R}\ p$.* ■

In the rest of this section we prove Proposition 5.7.1. Informally, we show that each probabilistic execution fragment H generated by an adversary of $Advs$ can be converted into two other probabilistic execution fragments H' and H'', each one generated by some adversary of $Advs_D$, such that $P_{H'}[e(H')] \leq P_H[e(H)] \leq P_{H''}[e(H'')]$. Then, if $\mathcal{R}$ is $\leq$ we use H'', and if $\mathcal{R}$ is $\geq$ we use H'.

An operation that is used heavily in the proof is called *deterministic reduction*. Let H be a probabilistic execution fragment of a probabilistic automaton M, and let q be a state of H. A probabilistic execution fragment H' is said to be obtained from H by deterministic reduction of the transition enabled from q if H' is obtained from H through the following two operations:

1. Let $tr_q^H = q \frown (\sum_{i \in I} p_i tr_i)$ where each p_i is non-zero and each tr_i is a transition of M. Then replace tr_q^H either with $(q, \mathcal{D}(\delta))$ or with $q \frown tr_j$, under the restriction that $(q, \mathcal{D}(\delta))$ can be chosen only if $\sum_{i \in I} p_i < 1$.

2. Remove all the states of H that become unreachable after tr_q^H is replaced.

Throughout the rest of this section we assume implicitly that whenever a probabilistic execution fragment is transformed, all the states that become unreachable are removed.

Lemma 5.7.2 *Let Advs be an execution-based adversary schema for a probabilistic automaton M, and let H be a probabilistic execution fragment of M that is generated by some adversary of Advs. Let e be an execution-based event schema such that $P_H[e(H)] = p$. Let q be a state of H. Then there exist two probabilistic execution fragments H^q_{low}, H^q_{high}, each one generated by an adversary of Advs, that are obtained from H by deterministic reduction of the transition enabled from q, and such that $P_{H^q_{low}}[e(H^q_{low})] \leq p$, and $P_{H^q_{high}}[e(H^q_{high})] \geq p$.*

Proof. Let tr^H_q be $q \frown (\sum_{i\in I} p_i tr_i)$, where each tr_i is either a transition of M or the pair $(lstate(q), \mathcal{D}(\delta))$, each p_i is greater than 0, and $\sum_{i\in I} p_i = 1$. For each transition tr_i, $i \in I$, let H_{tr_i} be obtained from H by replacing tr^H_q with $q \frown tr_i$. Observe that, since *Advs* is execution-based and H is generated by an adversary of *Advs*, H_{tr_i} is generated by an adversary of *Advs*. The probability of $e(H)$ can be written as

$$P_H[e(H)] = P_H[C_q]P_H[e(H)|C_q] + (1 - P_H[C_q])P_H[e(H)|\overline{C_q}]. \tag{5.47}$$

Observe that for each $i \in I$, since H and H_{tr_i} differ only in the states having q as a prefix, $P_H[C_q] = P_{H_{tr_i}}[C_q]$. Since e is execution-based, $e(H) \cap \overline{C_q} = e(H_{tr_i}) \cap \overline{C_q}$, and $P_H[e(H) \cap \overline{C_q}] = P_{H_{tr_i}}[e(H_{tr_i}) \cap \overline{C_q}]$ (use conditional probability spaces and Theorem 3.1.2). Moreover, as it is shown below, $P_H[e(H) \cap C_q] = \sum_{i\in I} p_i P_{H_{tr_i}}[e(H_{tr_i}) \cap C_q]$. In fact,

$$P_H[e(H)\cap C_q] = P_H[C_q]\left(P^H_q[\delta]P_H[e(H)|C_{q\delta}] + \sum_{(a,q')\in\Omega^H_q} P^H_q[(a,q')]P_H[e(H)|C_{q'}]\right), \tag{5.48}$$

where we assume that $P_H[e(H)|C_{q\delta}]$ is 0 whenever it is undefined. For each (a, q') of Ω^H_q, $P^H_q[(q, a')] = \sum_{i\in I} p_i P^{H_{tr_i}}_q[(a, q')]$, and for each i such that $(a, q') \in \Omega^{H_{tr_i}}_q$, $P_H[e(H)|C_{q'}] = P_{H_{tr_i}}[e(H_{tr_i})|C_{q'}]$ (simply observe that $H \triangleright q' = H_{tr_i} \triangleright q'$). Similarly, if $\delta \in \Omega^H_q$, then $P^H_q[\delta] = \sum_{i\in I} p_i P^{H_{tr_i}}_q[\delta]$, and for each i such that $\delta \in \Omega^{H_{tr_i}}_q$, $P_H[e(H)|C_{q\delta}] = P_{H_{tr_i}}[e(H_{tr_i})|C_{q\delta}]$. Thus, from (5.48),

$$P_H[e(H) \cap C_q] = \sum_{i\in I} p_i P_{H_{tr_i}}[C_q]$$
$$\left(P^{H_{tr_i}}_q[\delta]P_{H_{tr_i}}[e(H_{tr_i})|C_{q\delta}] + \sum_{(a,q')\in\Omega^{H_{tr_i}}_q} P^{H_{tr_i}}_q[(a,q')]P_{H_{tr_i}}[e(H_{tr_i})|C_{q'}]\right), \tag{5.49}$$

which gives the desired equality

$$P_H[e(H) \cap C_q] = \sum_{i\in I} p_i P_{H_{tr_i}}[e(H_{tr_i}) \cap C_q]. \tag{5.50}$$

Thus, (5.47) can be rewritten into

$$P_H[e(H)] = \sum_{i\in I} p_i \left(P_{H_{tr_i}}[C_q]P_{H_{tr_i}}[e(H_{tr_i})|C_q] + (1 - P_{H_{tr_i}}[C_q])P_{H_{tr_i}}[e(H_{tr_i})|\overline{C_q}]\right), \tag{5.51}$$

which becomes

$$P_H[e(H)] = \sum_{i\in I} p_i P_{H_{tr_i}}[e(H_{tr_i})]. \tag{5.52}$$

If there exists an element i of I such that $P_{H_{tr_i}}[e(H_{tr_i})] = p$, then fix H^q_{low} and H^q_{high} to be H_{tr_i}. If there is no element i of I such that $P_{H_{tr_q}}[e(H_{tr_i})] = p$, then it is enough to show that there are two elements i_1, i_2 of I such that $P_{H_{tr_{i_1}}}[e(H_{tr_{i_1}})] < p$ and $P_{H_{tr_{i_2}}}[e(H_{tr_{i_2}})] > p$, respectively. Assume by contradiction that for each element i of I, $P_{H_{tr_i}}[e(H_{tr_i})] < p$. Then, from (5.52), $\sum_{i \in I} p_i P_{H_{tr_i}}[e(H_{tr_i})] < p$, which contradicts $P_H[e(H)] = p$. Similarly, assume by contradiction that for each element i of I, $P_{H_{tr_i}}[e(H_{tr_i})] > p$. Then, from (5.52), $\sum_{i \in I} p_i P_{H_{tr_i}}[e(H_{tr_i})] > p$, which contradicts $P_H[e(H)] = p$ again. ■

Lemma 5.7.3 *Let Advs be an execution-based adversary schema for a probabilistic automaton M, and let H be a probabilistic execution fragment of M that is generated by some adversary of Advs. Let e be an execution-based event schema such that $P_H[e(H)] = p$. Let d be a natural number, and let U_d be the set of states q of H such that $|q| = d$. Then there exist two probabilistic execution fragments H_{low}, H_{high}, each one generated by an adversary of Advs, that are obtained from H by deterministic reduction of the transitions enabled from the states of U_d, and such that $P_{H_{low}}[e(H_{low})] \leq p$, and $P_{H_{high}}[e(H_{high})] \geq p$.*

Proof. From Lemma 5.7.2 we know that for each state q of U_d there are two probabilistic execution fragments H^q_{low} and H^q_{high}, obtained from H by deterministic reduction of the transition enabled from q, such that $P_{H^q_{low}}[e(H^q_{low})] \leq p$, and $P_{H^q_{high}}[e(H^q_{high})] \geq p$. Let H_{low} be obtained from H by replacing the transition enabled from each state q of U_d with the transition enabled from q in H^q_{low}, and let H_{high} be obtained from H by replacing the transition enabled from each state q of U_d with the transition enabled from q in H^q_{high}. Since $Advs$ is execution-based and all the involved probabilistic execution fragments are generated by an adversary of $Advs$, then H_{high} and H_{low} are generated by an adversary of $Advs$. Since e is execution-based, for each state q of U_d, $P_{H_{low}}[e(H_{low}) \cap C_q] = P_{H^q_{low}}[e(H^q_{low}) \cap C_q]$. Thus,

$$P_{H_{low}}[e(H_{low})] = \sum_{q \in U_d} P_{H_{low}}[C_q] P_{H^q_{low}}[e(H^q_{low}) | C_q]. \tag{5.53}$$

Observe that, for each state q of U_d, the difference between the probability of $e(H)$ and the probability of $e(H^q_{low})$ is determined by the subcones of C_q. Thus,

$$P_{H_{low}}[e(H_{low})] \leq \sum_{q \in U_d} P_H[C_q] P_H[e(H) | C_q]. \tag{5.54}$$

The right side of (5.54) is $P_H[e(H)]$, which is p. In a similar way it is possible to show that $P_{H_{high}}[e(H_{high})] \geq p$. ■

Now we use the fact that e is finitely satisfiable. For each probabilistic execution fragment H of M, let $Can(e(H))$ the set of minimal elements of $\{q \in states(H) \mid C_q \subseteq e(H)\} \cup \{q\delta \mid q \in states(H), C_{q\delta} \subseteq e(H)\}$. Then, $Can(e(H))$ is a characterization of $e(H)$ as a union of disjoint cones. For each natural number d, let $e \restriction d$ be the function that given a probabilistic execution fragment H returns the set $\cup_{q \in Can(e(H)) || q | \leq d} C^H_q$.

Lemma 5.7.4 *Let e be an execution-based, finitely satisfiable, event schema for a probabilistic automaton M, and let d, d' be two natural numbers such that $d \leq d'$. Then, for each probabilistic execution fragment H, $P_H[e \restriction d(H)] \leq P_H[e \restriction d'(H)] \leq P_H[e(H)]$.*

Proof. Follows trivially from the definitions. ■

Lemma 5.7.5 *Let e be an execution-based, finitely satisfiable, event schema for a probabilistic automaton M, and let d be a natural number. Let H be a probabilistic execution fragment H of M, and let H' be obtained from H by reducing deterministically any collection of states of length greater than d. Then, $P_H[e\uparrow d(H)] \leq P_{H'}[e\uparrow d(H')]$.*

Proof. Just observe that for each $q \in Can(e(H))$ such that $|q| \leq d$ there is a $q' \in Can(e(H'))$ such that $q' \leq q$, and that for each state q of H such that $|q| \leq d$, $P_H[C_q] = P_{H'}[C_q]$. ■

Lemma 5.7.6 *Let $Advs$ be an execution-based adversary schema for a probabilistic automaton M, and let H be a probabilistic execution fragment of M that is generated by some adversary of $Advs$. Let e be an execution-based, finitely satisfiable event schema such that $P_H[e(H)] = p$. Then there exists a probabilistic execution fragment H', generated by a deterministic adversary of $Advs$, such that $P_{H'}[e(H')] \leq p$.*

Proof. From Lemma 5.7.3 it is possible to find a sequence of probabilistic execution fragments $(H_i)_{i\geq 0}$, where $H_0 = H$, each H_{i+1} is obtained from H_i by deterministically reducing all its transitions leaving from states of length i, and for each i, $P_{H_{i+1}}[e(H_{i+1})] \leq P_{H_i}[e(H_i)]$. Let H' be obtained from H by replacing the transition enabled from each state q with the transition enabled from q in any H_i such that $|q| \leq i$. It is immediate to check that H' is generated by some deterministic adversary of $Advs$ (every extended execution of $\Omega_{H'}$ is an extended execution of Ω_H).

Suppose by contradiction that $P_{H'}[e(H')] > p$. Then there exists a level d such that

$$P_{H'}[e\uparrow d(H')] > p. \tag{5.55}$$

For each $d' \geq d$, let $E_{d'}$ be

$$E_{d'} \stackrel{\triangle}{=} \bigcup_{q \in Can(e\uparrow d'(H_{d'}))\mid \exists_{q' \in Can(e\uparrow d(H'))} q' \leq q} C_q^{H'}. \tag{5.56}$$

Then, the following properties are valid.

1. for each $d' \geq d$, E'_d is an element of $\mathcal{F}_{H'}$.

   $E_{d'}$ is a union of cones of $\mathcal{F}_{H'}$.

2. if $d' \leq d''$, then $E_{d'} \subseteq E_{d''}$

   Consider an element $q \in Can(e\uparrow d'(H_{d'}))$ such that there exists a $q' \in Can(e\uparrow d(H'))$ such that $q' \leq q$. Observe that, since $H_{d''}$ is obtained from $H_{d'}$ by deterministic reduction of states of length greater than d', there exists a $q'' \in Can(e\uparrow d''(H_{d''}))$ such that $q'' \leq q$. Moreover, from the construction of H', $q' \leq q''$. Thus, from (5.56), $C_{q''}^{H'} \subseteq E_{d''}$. Since $q'' \leq q$, $C_q^{H'} \subseteq E_{d''}$, and therefore, $E_{d'} \subseteq E_{d''}$.

3. $e\uparrow d(H') \subseteq \cup_{d' \geq d} E_{d'}$.

   Consider an element α of $e\uparrow d(H')$. Then, for each d', $\alpha \in e(H_{d'})$. Let $q' \in Can(e(H_d))$ such that $q' \leq \alpha$, and let d' be $|q'|$. Then, there exists a $q'' \in Can(e\uparrow d'(H_{d'}))$ such that $q'' \leq q' \leq \alpha$, and thus $\alpha \in E_{d'}$.

4. for each $d' \geq d$, $P_{H_{d'}}[e \uparrow d'(H_{d'})] \geq P_{H'}[E_{d'}]$.

   From the construction of H', for each q such that $|q| \leq d'$, $P_{H_{d'}}[C_q^{H_{d'}}] = P_{H'}[C_q^{H'}]$. Moreover, if $C_q^{H'}$ is used in the definition of $E_{d'}$, then $q \in Can(e \uparrow d'(H_{d'}))$.

From 2 and 3, and from (5.55), there exists a value d' such that $P_{H'}[E_{d'}] > p$. From 4, $P_{H_{d'}}[e \uparrow d'(H_{d'})] > p$. From Lemma 5.7.4, $P_{H_{d'}}[e(H_{d'})] > p$. This contradicts the fact that $P_{H_{d'}}[e \uparrow d'(H_{d'})] \leq p$. ■

To build a probabilistic execution fragment H', generated by an adversary of $Advs_D$, such that $P_{H'}[e(H')] \geq p$, we need to extend part of Lemmas 5.7.2 and 5.7.3.

Lemma 5.7.7 *Let Advs be an execution-based adversary schema for a probabilistic automaton M, and let H be a probabilistic execution fragment of M that is generated by some adversary of Advs. Let e be an execution-based, finitely-satisfiable, event schema. Let q be a state of H, and let d be a natural number such that $P_H[e \uparrow d(H)] = p$. Then there exist a probabilistic execution fragment H^q_{high}, generated by an adversary of Advs, that is obtained from H by deterministic reduction of the transition enabled from q, such that $P_{H^q_{high}}[e \uparrow d(H^q_{high})] \geq p$.*

Proof. This proof is similar to the proof of Lemma 5.7.2, with the difference that the = sign of Equations (5.49), (5.50), (5.51), and (5.52), is changed into a $\leq$. In fact, in each one of the H_{tr_i} some new cone of length at most d may appear. ■

Lemma 5.7.8 *Let Advs be an execution-based adversary schema for a probabilistic automaton M, and let H be a probabilistic execution fragment of M that is generated by some adversary of Advs. Let e be an execution-based, finitely-satisfiable, event schema, and let d be a natural number such that $P_H[e \uparrow d(H)] = p$. Let d' be a natural number, and let $U_{d'}$ be the set of states q of H such that $|q| = d'$. Then there exist a probabilistic execution fragment H_{high}, generated by an adversary of Advs, that differs from H only in that the transitions enabled from the states of U_d are deterministically reduced, such that $P_{H_{high}}[e \uparrow d(H_{high})] \geq p$.*

Proof. This proof is similar to the proof of Lemma 5.7.3. In this case the arguments for the equation corresponding to Equation (5.54) is justified from the additional fact that H_{high} may have more cone of depth at most d than H. ■

Lemma 5.7.9 *Let Advs be an execution-based adversary schema for a probabilistic automaton M, and let H be a probabilistic execution fragment of M that is generated by some adversary of Advs. Let e be an execution-based, finitely-satisfiable, event schema such that $P_H[e(H)] > p$. Then, there exists a probabilistic execution fragment H' of M, generated by a deterministic adversary of Advs, such that $P_H[e(H')] > p$.*

Proof. Since $P_H[e(H)] > p$ and $e(H)$ is a union of cones, there exists a natural number d such that $P_H[e \uparrow d(H)] > p$. From repeated applications of Lemma 5.7.8, one for each level $d' \leq d$, there exists a probabilistic execution fragment H'', obtained from H by deterministic reduction of the transitions enabled from every state q with $|q| \leq d$, such that $P_{H''}[e \uparrow d(H'')] > p$. From Lemma 5.7.4, $P_{H''}[e(H'')] > p$. Moreover, any probabilistic execution fragment H''' obtained

from H'' by reducing deterministically transitions at depth greater than d ($|q| > d$) satisfies $P_{H'''}[e \restriction d(H''')] > p$, and thus $P_{H'''}[e(H''')] > p$. Hence, H' can be any probabilistic execution fragment obtained from H'' by reducing deterministically all the transitions at depth greater than d in any arbitrary way. It is easy to check that H' is generated by a deterministic adversary of $Advs$. ■

Lemma 5.7.10 *Let $Advs$ be an execution-based adversary schema for a probabilistic automaton M, and let H be a probabilistic execution fragment of M that is generated by some adversary of $Advs$. Let e be an execution-based, finitely-satisfiable, event schema such that $P_H[e(H)] \geq p$. Then, there exists a probabilistic execution fragment H' of M, generated by a deterministic adversary of $Advs$, such that $P_H[e(H')] \geq p$.*

Proof. If $P_H[e(H)] > p$, then Lemma 5.7.9 suffices. If $P_H[e(H)] = p$, then by Lemma 5.7.3 it is possible to find a sequence of probabilistic execution fragments $(H_i)_{i \geq 0}$, where $H_0 = H$, each H_{i+1} is obtained from H_i by deterministically reducing all its i-level transitions, and for each i, $P_{H_{i+1}}[e(H_{i+1})] \geq P_{H_i}[e(H_i)]$. If there exists a sequence $(H_i)_{i \geq 0}$ such that for some i, $P_{H_i}[e(H_i)] > p$, then Lemma 5.7.9 suffices. Otherwise, consider the sequence of probabilistic execution fragments defined as follows: $H_0 = H$ and, for each i, let d_i be the level of H_i such that $P_{H_i}[e \restriction d_i(H_i)] \geq p \sum_{j \leq i} (1/2)^{j+1}$. Let H_{i+1} be obtained from repeated applications of Lemma 5.7.8, till level d_i, so that $P_{H_{i+1}}[e \restriction d_i(H_{i+1})] \geq p \sum_{j \leq i} (1/2)^{j+1}$. Note that $P_{H_{i+1}}[e(H_{i+1})] = p$, otherwise we can find a sequence $(H_i)_{i \geq 0}$ and an i such that $P_{H_{i+1}}[e(H_{i+1})] > p$ (simple argument by contradiction). Let H' be obtained from H by replacing the transition enabled from each state q with the transition enabled from q in any H_i such that $|q| \leq d_{i-1}$. It is easy to check that H' is generated by an adversary of $Advs$. Suppose by contradiction that $P_{H'}[e(H')] = p' < p$. Then, from the construction of the H_i's, there exists an i such that $p \sum_{j \leq i} (1/2)^{j+1} > p'$, and thus $P_{H_{i+1}}[e \restriction d_i(H_{i+1})] > p'$. However, from the definition of H', $P_{H_{i+1}}[e \restriction d_i(H_{i+1})] = P_{H'}[e \restriction d_i(H')]$, and thus $p' < P_{H'}[e(H')]$, which contradicts the fact that $P_{H'}[e(H')] = p'$. ■

Proof of Proposition 5.7.1. Since $Advs_D \subseteq Advs$, $\Pr_{Advs,\Theta}(e)\ \mathcal{R}\ p$ implies $\Pr_{Advs_D,\Theta}(e)\ \mathcal{R}\ p$ trivially. Conversely, suppose that $\Pr_{Advs_D,\Theta}(e)\ \mathcal{R}\ p$, and let H be a probabilistic execution fragment, generated by an adversary of $Advs$, whose start state is in Θ. We distinguish the following cases.

1. $\mathcal{R}$ is $\geq$.

   From Lemma 5.7.6, there is a probabilistic execution fragment H', generated by an adversary of $Advs_D$, whose start state is in Θ, and such that $P_{H'}[e(H')] \leq P_H[e(H)]$. From hypothesis, $P_{H'}[e(H')] \geq p$. Thus, $P_H[e(H)] \geq p$.

2. $\mathcal{R}$ is $\leq$.

   From Lemma 5.7.10, there is a probabilistic execution fragment H', generated by an adversary of $Advs_D$, whose start state is in Θ, and such that $P_{H'}[e(H')] \geq P_H[e(H)]$. From hypothesis, $P_{H'}[e(H')] \leq p$. Thus, $P_H[e(H)] \leq p$.

3. $\mathcal{R}$ is $=$.

   This follows by combining Items 1 and 2. ■

5.7.2 Execution-Based Adversary Schemas with Partial On-Line Information

Proposition 5.7.1 can be extended to adversary schemas that do not know all the past history of a system, i.e., to execution-based adversary schemas with partial on-line information. We need to impose a technical restriction, though, which is that an adversary should always be able to distinguish two execution fragments with a different length (cf. Example 5.7.2). The proof of the new result is a simple modification of the proof of Proposition 5.7.1.

Proposition 5.7.11 *Let $(\equiv, F)$ be an oblivious relation such that for each pair $\alpha_1 \equiv \alpha_2$ of equivalent execution fragment, α_1 and α_2 have the same length. Let Advs be an execution-based adversary schema with partial on-line information such that each adversary of Advs is oblivious relative to $(\equiv, F)$, and let $Advs_D$ be the set of deterministic adversaries of Advs. Let e be a finitely-satisfiable, execution-based, event schema for M. Then, for every set Θ of finite execution fragments of M, every probability p, and every relation $\mathcal{R}$ among $\leq$, $=$, $\geq$, $\Pr_{Advs,\Theta}(e)\ \mathcal{R}\ p$ iff $\Pr_{Advs_D,\Theta}(e)\ \mathcal{R}\ p$.*

Proof. The proof is similar to the proof of Proposition 5.7.1. The main difference is in the proofs of Lemmas 5.7.2, 5.7.3 and 5.7.8, where equivalence classes of states rather than single states only must be considered. In these two proofs we use also the fact that equivalent execution fragments have the same length. The details of the proof are left to the reader. ■

Example 5.7.2 (Why length sensitivity) The requirement that an adversary should always see the length of a probabilistic execution fragment seems to be artificial; however, randomized adversaries have more power in general if they cannot see the length of a probabilistic execution. Consider the probabilistic automaton M of Figure 5-3, and suppose that all the executions of M that end in states s_1, s_2, s_3, and s_6 are equivalent. Since for each state s_i there is exactly one execution of M that ends in s_i, we denote such an execution by q_i. Let Θ be the set of extended executions $\alpha\delta$ of M such that $lstate(\alpha)$ does not enable any transition in M. For each state s_i that enables some transition, let $tr_{i,u}$ be the transition that leaves from s_i and goes upward, and let $tr_{i,d}$ be the transition that leaves from s_i and goes downward. Then, for each pair $i, j \in \{1,2,3,6\}$, $i \neq j$, let $f_{q_i q_j}(tr_{i,u}) = tr_{j,u}$, and let $f_{q_i q_j}(tr_{i,d}) = tr_{j,d}$.

Let $Advs$ be the set of Θ-based adversaries for M that are oblivious relative to $(\equiv, F)$, and let $Advs_D$ be the set of deterministic adversaries of $Advs$. Then, the statement $\{s_0\} \xrightarrow[1/2]{} Advs_D \{s_7, s_{10}\}$ is valid, whereas the statement $\{s_0\} \xrightarrow[1/2]{} Advs \{s_7, s_{10}\}$ is not valid, i.e., an adversary can use randomization to reduce the probability to reach states $\{s_7, s_{10}\}$. In fact, the probabilistic executions H_1 and H_2 of Figure 5-3 are the only probabilistic executions of M that can be generated by the adversaries of $Advs_D$, while H_0 is generated by an adversary of $Advs$. The probability of reaching $\{s_7, s_{10}\}$ in H_1 and H_2 is $1/2$, whereas the probability of reaching $\{s_7, s_{10}\}$ in H_0 is $1/4$. ■

5.8 Probabilistic Statements without Adversaries

The current literature on randomized distributed algorithms relies on the notion of an adversary, and for this reason all the definitions given in this chapter are based on adversaries. However,

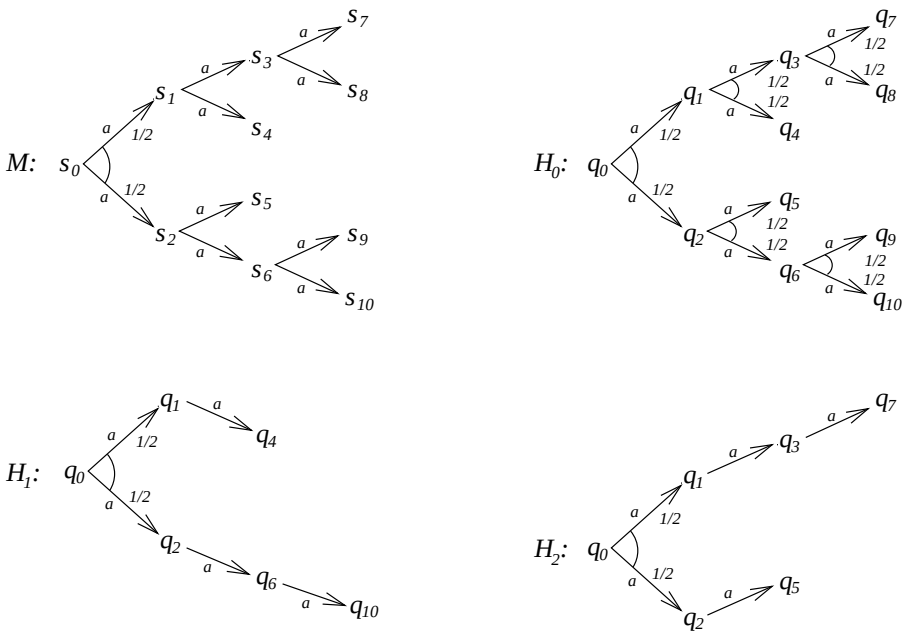


Figure 5-3: Randomization adds power for some adversaries with partial on-line information.

the key objects of the theory that we have presented are the probabilistic execution fragments of a probabilistic automaton, and not its adversaries. An adversary schema can be replaced by an arbitrary set of probabilistic execution fragments in the definition of a probabilistic statement, namely, the set of probabilistic execution fragments that the adversary schema can generate. In other words, an adversary schema can be seen as a useful tool to express a set of probabilistic execution fragments.

5.9 Discussion

Two objects that we have defined in this chapter and that do not appear anywhere in the literature are adversary schemas and event schemas. Both the objects are needed because, differently from existing work, in this thesis we identify several different rules to limit the power of an adversary and several different rules to associate an event with a probabilistic execution fragment, and thus we need some way to identify each rule. The best way to think of an adversary schema and of an event schema is as a way to denote the rule that is used to limit the power of an adversary and denote the rule that is used to associate an event with each probabilistic execution fragment.

We have defined the classes of execution-based adversary schemas and execution-based event schemas, and we have proved that for finitely satisfiable execution-based event schemas randomization does not increase the power of an execution-based adversary schema, or of a class of execution-based adversary schemas with partial on-line information. These results are of practical importance because most of the known event schemas and adversary schemas of practical interest are execution-based. As a result, it is possible to verify the correctness of a randomized distributed algorithm by analyzing only the effect of deterministic adversaries,

which is easier than analyzing every adversary. A similar result is shown by Hart, Sharir and Pnueli [HSP83] for fair adversaries and almost-sure termination properties, i.e., properties that express the fact that under all fair adversaries the system reaches some fixed set of states with probability 1. Fair adversaries and termination events are expressible as execution-based adversary schemas and finitely satisfiable execution-based event schemas, respectively; thus, the result of Hart, Sharir and Pnueli is implied by our result. Hart, Sharir and Pnueli prove also that another class of adversaries is equivalent to the class of fair adversaries, namely, those adversaries that lead to fair executions with probability 1. The same result holds here as well; however, it is not clear under what conditions a similar result holds in general.