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

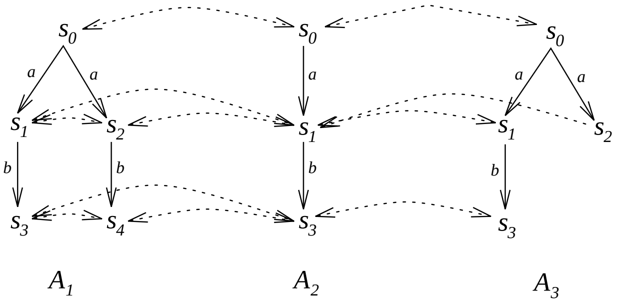


Figure 1-1: Simulation relations for automata.

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

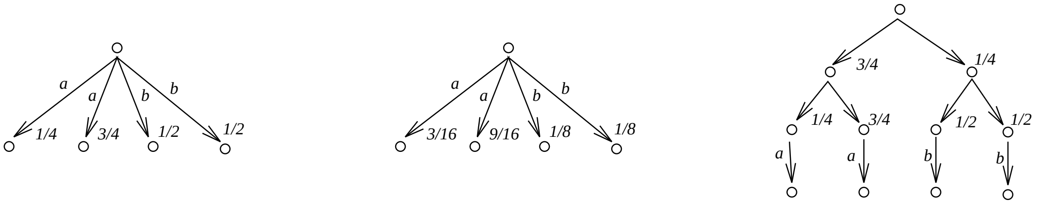


Figure 2-1: Reactive, generative and stratified processes, from left to right.

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 probability 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 resembles 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 $f : \Omega \rightarrow \Omega'$ 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 $\Omega_{exp}, \mathcal{F}_{exp}$, and P_{exp} to denote $\Omega, \mathcal{F}$, 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*.

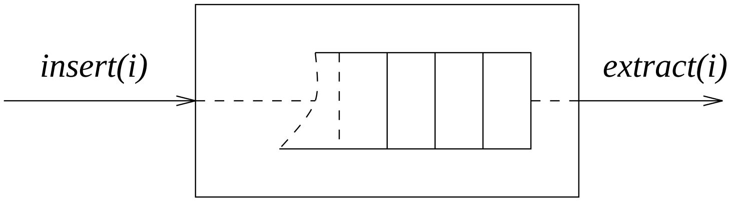


Figure 3-1: The *Buffer* automaton.

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 $s \overset{a}{\Longrightarrow} s'$. The action a must be external, and the meaning of $s \overset{a}{\Longrightarrow} s'$ 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.

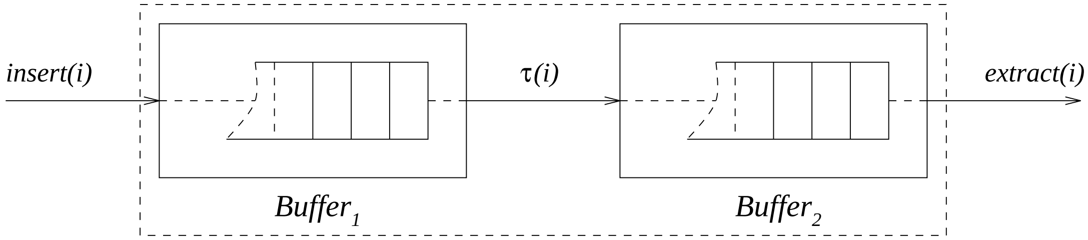


Figure 3-2: Concatenation of two buffers.

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 $\mathit{fstate}(\alpha)$ the first state of α and, if α is finite, denote by $\mathit{lstate}(\alpha)$ the last state of α. Furthermore, denote by $\mathit{frag}^*(A)$ and $\mathit{frag}(A)$ 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 $\mathit{exec}^*(A)$ and $\mathit{exec}(A)$ 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., $trace(\alpha) = \alpha \upharpoonright ext(A)$. 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:

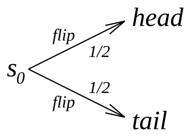


Figure 4-1: The coin flipping process.

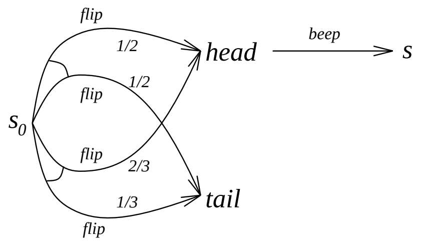


Figure 4-2: The nondeterministic coin flipping process.

> "*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

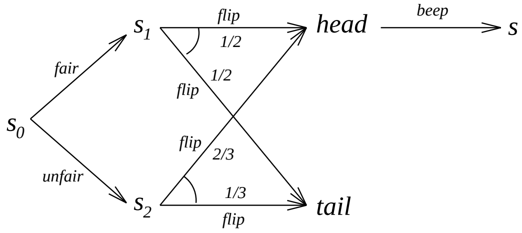


Figure 4-3: The triggered coin flipping process.

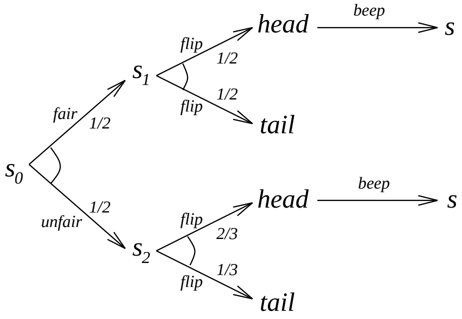


Figure 4-4: A computation of the triggered coin flipping process.

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 $\mathit{trans}(M)$ there is an action a of M such that $\Omega \subseteq \{a\} \times \mathit{states}(M)$. In such a case, a transition can be represented alternatively as $(s, a, \mathcal{P}')$, where $\mathcal{P}' \in \mathit{Probs}(\mathit{states}(M))$, 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,

$$\alpha = s_0a_1s_1a_2s_2\cdots,$$

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. ∎

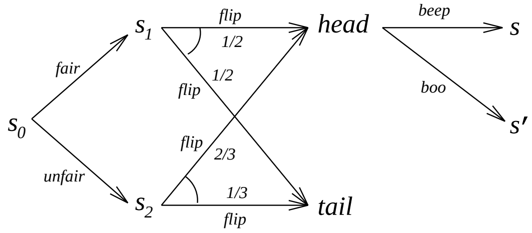


Figure 4-5: The triggered coin flipper with a *boo* sound.

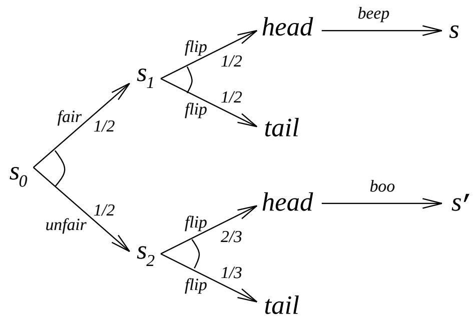


Figure 4-6: A computation of the triggered coin flipper with a *boo* sound.

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

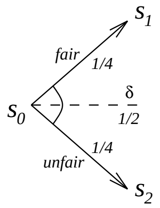


Figure 4-7: A probabilistic execution of the triggered coin flipper.

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 $s_0 \mathit{fair}\, s_1\delta$ 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

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

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

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

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.

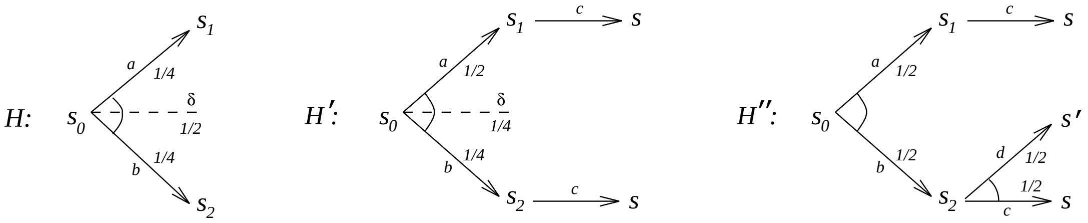


Figure 4-8: Examples of the prefix relation.

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.

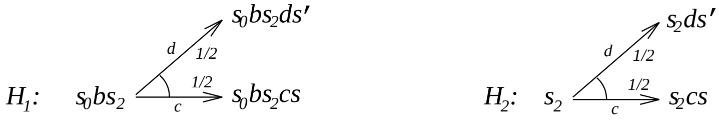


Figure 4-9: Conditionals and suffixes.

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 $H|q$, $tr_{q'}^{H|q} = tr_{q'}^{H}$.

$H|q$ 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 $s \overset{a}{\Longrightarrow} \mathcal{P}$ 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, $s \overset{a}{\Longrightarrow} \mathcal{P}$, 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 $\alpha\delta \in \Omega_H$, $trace(\alpha) = a$;

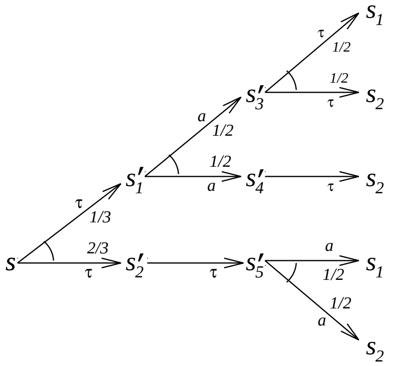


Figure 4-10: A representation of a weak transition with action a.

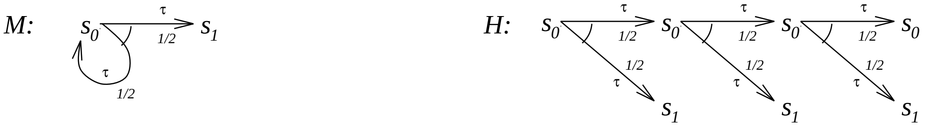


Figure 4-11: A weak transition of a probabilistic automaton with cycles.

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, $s \overset{a}{\Longrightarrow}_C \mathcal{P}$, is defined as a weak transition by dropping Condition 5. Throughout the thesis we also the extend the function δ-*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 $\mathcal{O}(\alpha) = (\mathit{lstate}(\alpha), \mathcal{D}(\delta))$ 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 $(\mathit{lstate}(q), \mathcal{D}(\delta))$ 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 $s \overset{a \upharpoonright ext(M)}{\Longrightarrow} \mathcal{P}$ 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 $s' = \mathit{lstate}(\alpha)$. Call α *active* if $\mathcal{P} \neq \mathcal{D}(\delta)$.

2. If $\alpha b s'$ is active, then $\mathit{fstate}(\alpha) = s$ 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 $a \upharpoonright ext(M)$.

4. For each reachable execution fragment $\alpha = s a_1 s_1 a_2 s_2 \cdots a_k s_k$, let

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

Then,

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

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

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

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 $P^{\mathcal{O}}_{\alpha}$ 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 $s \overset{a}{\Longrightarrow} \mathcal{P}$ 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

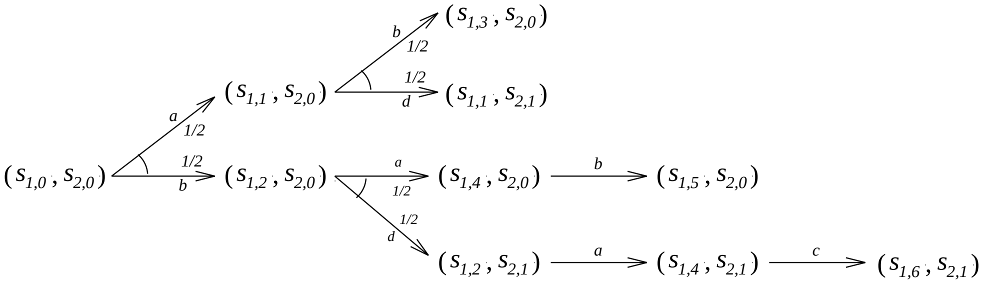


Figure 4-12: A probabilistic execution fragment of $M_1 \| M_2$.

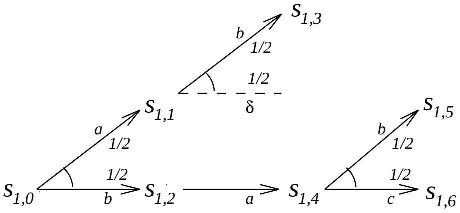


Figure 4-13: The projection onto M_1 of the probabilistic execution fragment of Figure 4-10.

(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.

$\boldsymbol{s_{1,4}}$ 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.

$\boldsymbol{s_{1,1}}$ 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 $tr \restriction V$, 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,*

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

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*

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

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

Proof. Let

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

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

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

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,

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

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

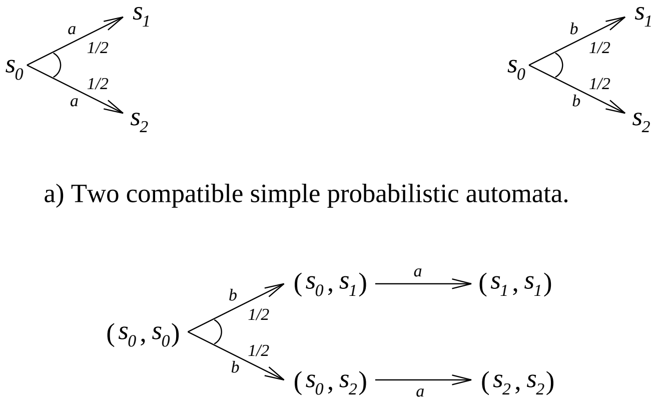
$$tr_q^{H'} \triangleq \sum_{q' \in q\rceil H} \bar{p}_{q'}^{q\rceil H} P_{q'}^H[acts(M_i)](tr_{q'}^H \upharpoonright acts(M_i))\lceil M_i. \tag{4.22}$$

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

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

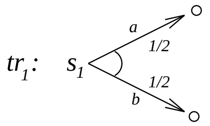
which is what we needed to show. ■

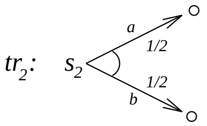
4.3.3 Parallel Composition for General Probabilistic Automata

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.

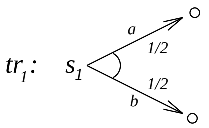
   

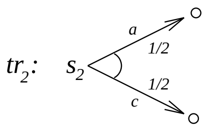
   If we choose not to synchronize tr_1 and tr_2, then the only transitions that can be synchronized are the simple transitions, leading to a trivial parallel composition operator that does not handle any kind of transition with probabilistic choices over actions. The transitions tr_1 and tr_2 cannot be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



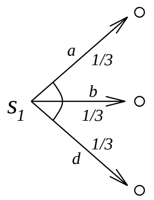


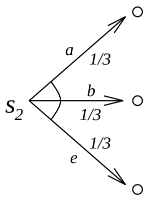
Note that actions a, b and c are all in common between M_1 and M_2. If we choose not to synchronize tr_1 and tr_2, then only transitions involving the same sets of actions can synchronize. However, we have the same problem outlined in Case 1, where neither tr_1, nor tr_2 can be scheduled independently.

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.

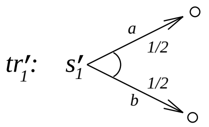
3. Suppose that from state s_1 of M_1 there is a transition tr_1 giving actions a, b, d probability $1/3$ to occur, and suppose that from state s_2 of M_2 there is a transition tr_2 giving actions a, b, e probability $1/3$ to occur.

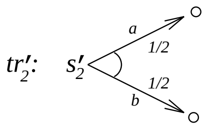




In this case each transition has some actions that are in common between M_1 and M_2, and some actions that are not in common.

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

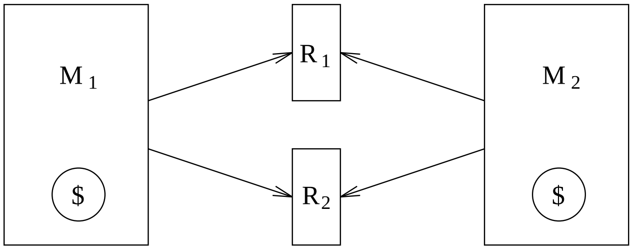


Figure 5-1: A toy resource allocation protocol.

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*(α). Formally,

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

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

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 $\mathit{Advs}(M)$ and $\mathit{DAdvs}(M)$, 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 $\mathit{Advs}(M)$. We call a subset of $\mathit{Advs}(M)$ 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 $\mathit{prexec}(M, \mathcal{A}, \alpha)$, is the unique probabilistic execution fragment H of M such that

1. $\mathit{start}(H) = \{\alpha\}$, 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 $\mathit{prexec}(M, \mathcal{A}, H)$, is the unique probabilistic execution fragment H' of M such that

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

$$p\left(\mathit{tr}_q^H \restriction \mathit{acts}(H)\right) + (1-p)\left(q \frown \mathcal{A}(q)\right),$$

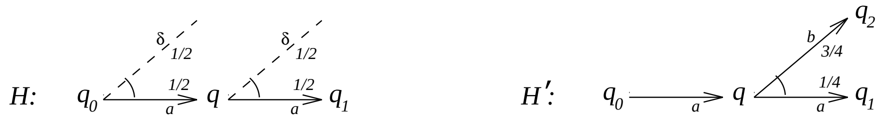


Figure 5-2: An example of the action of an adversary on a probabilistic execution fragment.

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,

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

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_1p_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

$$U \underset{p}{\longrightarrow}_{Advs} U', \tag{5.23}$$

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 $\mathit{fstate}(\alpha') = \mathit{lstate}(\alpha)$, $\mathcal{A}'(\alpha') = \mathcal{A}(\alpha \frown \alpha')$. 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 $\mathit{fstate}(q') = \mathit{lstate}(q)$, $\mathcal{A}_q(q') = \mathcal{A}(q \frown q')$. We know that $\mathcal{A}_q$ exists since *Advs* is finite-history-insensitive. It is a simple inductive argument to show that

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

Moreover,

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

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

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

Thus, from Proposition 4.2.12 and (5.27),

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

From hypothesis,

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

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,*

$$U \underset{p}{\longrightarrow}_{Advs} U' \text{ and } U' \underset{p'}{\longrightarrow}_{Advs} U'' \text{ imply } U \underset{pp'}{\longrightarrow}_{Advs} U''.$$

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

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

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

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

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),

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

From (5.32) and (5.34),

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

This shows that $U \xrightarrow[pp']{} {}_{Advs} U''$. ■

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

1. $U \xrightarrow[1]{} U$.

2. *If* $U_1 \xrightarrow[p_1]{} U_1'$ *and* $U_2 \xrightarrow[p_2]{} U_2'$, *then* $U_1 \cup U_2 \xrightarrow[min(p_1,p_2)]{} U_1' \cup U_2'$. ■

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

$$U \xrightarrow[p]{} {}_{Advs} V \tag{5.36}$$

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

$$\mathrm{Pr}_{Advs,U}(e_V) \geq p. \tag{5.37}$$

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

$$V \xrightarrow[p]{} {}_{Advs} U \tag{5.38}$$

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

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

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

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

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 $\mathit{prexec}(M, \mathcal{A}, q_0)$, 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 $q_0 \leq q$, $\mathit{lstate}(q) \in U$, 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 $U \underset{p}{\longrightarrow}_{Advs} X$, 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, $U \underset{1}{\longrightarrow}_{Advs} X$.*

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 $\leq$, $=$, $\geq$, $\mathrm{Pr}_{Advs,\Theta}(e)\ \mathcal{R}\ p$ iff $\mathrm{Pr}_{Advs_D,\Theta}(e)\ \mathcal{R}\ p$.* ■

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 $P_{H_{low}}[e(H_{low})] \le p$, and $P_{H_{high}}[e(H_{high})] \ge p$.*

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 $P_{H^q_{low}}[e(H^q_{low})] \le p$, and $P_{H^q_{high}}[e(H^q_{high})] \ge p$. 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,

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

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

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 $e\lceil d$ be the function that given a probabilistic execution fragment H returns the set $\cup_{q\in Can(e(H))||q|\le d} C^H_q$.

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 $d \le d'$. Then, for each probabilistic execution fragment H, $P_H[e\lceil d(H)] \le P_H[e\lceil d'(H)] \le P_H[e(H)]$.*

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, $P_H[e\lceil d(H)] \leq P_{H'}[e\lceil d(H')]$.*

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

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

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

$$E_{d'} \triangleq \bigcup_{q\in Can(e\lceil d'(H_{d'}))|\exists_{q'\in Can(e\lceil d(H'))} q'\leq q} C_q^{H'}. \tag{5.56}$$

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 $q \in Can(e\lceil d'(H_{d'}))$ such that there exists a $q' \in Can(e\lceil d(H'))$ 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 $q'' \in Can(e\lceil d''(H_{d''}))$ 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. $e\lceil d(H') \subseteq \cup_{d'\geq d} E_{d'}$.

   Consider an element α of $e\lceil d(H')$. 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 $q'' \in Can(e\lceil d'(H_{d'}))$ such that $q'' \leq q' \leq \alpha$, and thus $\alpha \in E_{d'}$.

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

   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 $q \in Can(e\lceil d'(H_{d'}))$.

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

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 $P_H[e\lceil d(H)] = p$. 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 $P_{H^q_{high}}[e\lceil d(H^q_{high})] \geq p$.*

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 $P_H[e\lceil d(H)] = p$. 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 $P_{H_{high}}[e\lceil d(H_{high})] \geq p$.*

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 $P_H[e\lceil d(H)] > p$. 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 $P_{H''}[e\lceil d(H'')] > p$. 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 $P_{H'''}[e\lceil d(H''')] > p$, 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 $P_{H_i}[e\lceil d_i(H_i)] \geq p\sum_{j\leq i}(1/2)^{j+1}$. Let H_{i+1} be obtained from repeated applications of Lemma 5.7.8, till level d_i, so that $P_{H_{i+1}}[e\lceil d_i(H_{i+1})] \geq p\sum_{j\leq i}(1/2)^{j+1}$. 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 $P_{H_{i+1}}[e\lceil d_i(H_{i+1})] > p'$. However, from the definition of H', $P_{H_{i+1}}[e\lceil d_i(H_{i+1})] = P_{H'}[e\lceil d_i(H')]$, and thus $p' < P_{H'}[e(H')]$, which contradicts the fact that $P_{H'}[e(H')] = p'$. ■

Proof of Proposition 5.7.1. Since $Advs_D \subseteq Advs$, $\mathrm{Pr}_{Advs,\Theta}(e)\;\mathcal{R}\;p$ implies $\mathrm{Pr}_{Advs_D,\Theta}(e)\;\mathcal{R}\;p$ trivially. Conversely, suppose that $\mathrm{Pr}_{Advs_D,\Theta}(e)\;\mathcal{R}\;p$, 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,

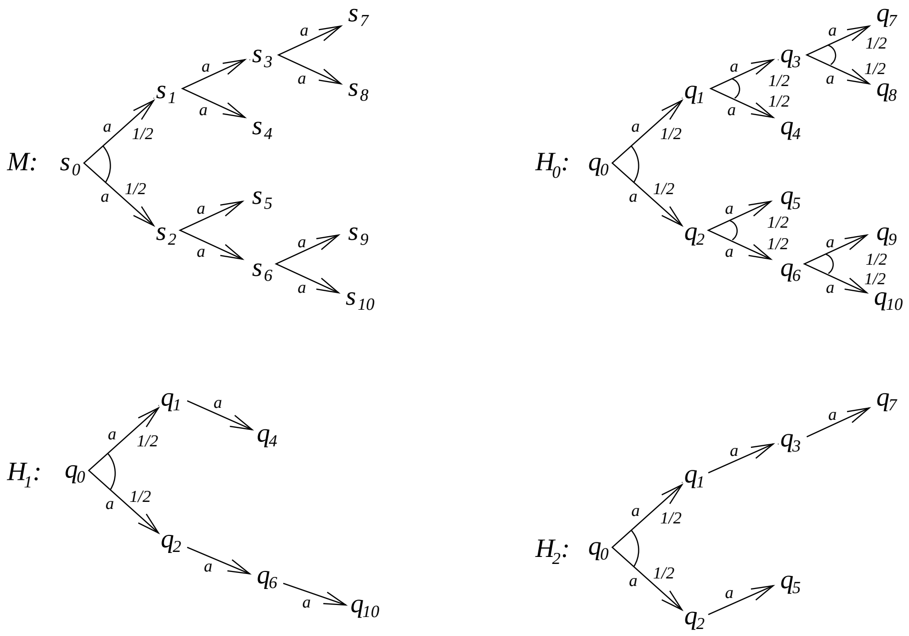


Figure 5-3: Randomization adds power for some adversaries with partial on-line information.

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.

Chapter 6

Direct Verification: Proving a Property

In this chapter we illustrate techniques to prove the validity of a probabilistic statement from scratch. The main technique, which is based on *coin lemmas*, consists of reducing the analysis of a property of a probabilistic automaton to the analysis of a property of an ordinary automaton. We illustrate the methodology by applying it to some existing randomized algorithms.

Part of this chapter is based on joint work with Anna Pogosyants and Isaac Saias. Anna Pogosyants suggested us the coin event OCC (Section 6.2.3) as a generalization of other less elegant coin events that we had in mind and collaborated on the verification of the randomized algorithm for agreement of Ben-Or (Section 6.5). The verification of the randomized dining philosophers algorithm of Lehmann and Rabin (Section 6.3) is based on joint work with Nancy Lynch and Isaac Saias [LSS94], and the verification of the randomized algorithm for agreement of Ben-Or is a formalization of a proof that appears in the book on distributed algorithms of Nancy Lynch [Lyn95].

6.1 How to Prove the Validity of a Probabilistic Statement

In Chapter 5 we have defined formally what is a probabilistic statement and we have shown how it is possible to combine probabilistic statements to derive more complex properties. However, one question is left open: how do we prove the validity of a given probabilistic statement from scratch?

The problem is not trivial: a property may rely on complicate global configurations of a system that depend on several separated random draws. Analyzing the exact probability of an event associated with a probabilistic execution fragment may be extremely hard. Fortunately, there are usually some key points, known to the designer of a system, where specific probabilistic choices lead to the desired property. In this chapter we formalize the idea above by introducing a collection of *coin lemmas*. The idea behind a coin lemma is the following.

1. We define a mechanism to identify events of the kind "some specific probabilistic choices yield some specific results". We call such events *coin events* since a common source of randomness is given by coin flips.

2. We prove a lower bound on the probability of the coin event that we identify.

Then, the analysis of a probabilistic statement for a probabilistic automaton M proceeds as follows.

1. We find a coin event that expresses the key intuition behind the property to be shown.
2. We show that the coin event is a subevent of the event expressing the desired property, i.e., we show that whenever the coin event is satisfied, the desired property is satisfied as well.
3. We use the lower bound on the probability of the coin event to obtain a lower bound on the probability of the desired property.

Example 6.1.1 (Coin lemmas and the toy resource allocation protocol) Let us consider the toy resource allocation protocol of Chapter 5 again. One of the coin lemmas of this chapter states that if we fix any two separate coin flips (flipping of different coins) and we consider the event where the two coin flips yield different outcomes whenever they both occur, then, no matter how the nondeterminism is resolved, the considered event is satisfied with probability at least 1/2. On the other hand, if the first coin flip of M_1 after the first coin flip of M_2 is different from the last coin flip of M_2 before the first time M_1 checks its resource after flipping, then M_1 succeeds in getting its resource. Thus, whenever the property above can be expressed as a *coin event* in a form suitable to the coin lemma above, we are guaranteed that M_1 eventually gets its resource with probability at least 1/2. It turns out that an adversary must be fair, oblivious and deterministic in order to be able to define the desired coin event (cf. Section 6.6). Our results about deterministic and randomized adversaries (Proposition 5.7.11) can then be used to remove the constraint that an adversary is deterministic. ■

We present a large collection of coin lemmas, and we illustrate their use via two main examples: Section 6.3 proves the correctness of the randomized Dining Philosophers algorithm of Lehmann and Rabin [LR81], and Section 6.5 proves the correctness of the randomized algorithm of Ben-Or for agreement in asynchronous networks in the presence of stopping faults [BO83]. At the end of the chapter we hint at another technique, called the *partition technique*, that departs considerably from the coin lemmas and that is necessary to prove stronger claims about the toy resource allocation protocol. We leave to further work a deeper study of this other technique.

6.2 Some Simple Coin Lemmas

In this section we present some simple coin lemmas where we use actions to identify the random draws of interest. Specifically, we study the following coin lemmas.

1. *First occurrence of an action.*

   In this coin lemma we consider an action a and a set of states U, and we study the probability that either action a does not occur or the first occurrence of action a leads to a state of U. We show that this probability is at least the infimum of the probability of reaching a state of U over all the transitions of M that are labeled with action a.

As an example, action a can identify the process of flipping a fair coin and U can identify those states that are reached if the coin flip yields head. Then the coin lemma says that no matter how the nondeterminism is resolved the probability that either the coin is not flipped or the coin is flipped and yields head is at least 1/2.

Observe that in the definition of the coin event we allow for those executions where no coin is flipped. One reason for this choice is to avoid trivial lower bounds due to the fact that a generic adversary can always decide not to schedule any transition. Another reason is that generally a randomized algorithm is structured so that that if no coin is flipped then progress is guaranteed with certainty. Alternatively, a randomized algorithm can be structured so that under any valid adversary some coin is flipped. In both cases it is of absolute importance to be aware of the existence of executions where no coin is flipped. Overlooking those executions is a common source of mistakes.

2. *First occurrence of an action among many.*

   In this coin lemma we consider several pairs (a_i, U_i) of actions and sets of states, and we study the probability that either none of the a_i's occur or the action a_j that occurs first leads to a state of U_j. We show that, if for each i p_i is the lower bound given for (a_i, U_i) by the coin lemma of 1, then the probability mentioned above is at least the minimum of the p_i's.

   As an example, consider n processes that run in parallel, and suppose that each process can flip a fair coin. Then, the probability that either no process flips a coin or that the first process that flips a coin obtains head is at least 1/2.

3. *I-th occurrence of an action among many.*

   In this coin lemma we consider the coin event of 2 with the difference that we consider the i^{th} occurrence of an action rather than the first occurrence. The lower bound on the probability of this event is the same as the lower bound on the probability of the event of 2.

4. *Conjunction of separate coin events.*

   In this coin lemma we consider the conjunction of several coin events of the kind of 3. We show that if each one of the coin events involves disjoint occurrences of actions, then the lower bound on the probability of the conjunction is the product of the lower bounds on the probability of each of the involved coin events.

   As an example, consider n processes that run in parallel, and suppose that each process can flip a fair coin. For each i let x_i be either head or tail. Then, the probability that for each process i either no coin is flipped or the first coin that is flipped yields x_i is at least $1/2^n$.

Some more general and complex coin lemmas are presented in Section 6.4; several other coin lemmas are likely to be derived in the future. Before presenting the simple coin lemmas in full detail we give just a rough idea of the coin lemmas of Section 6.4.

5. *Conjunction of separate coin events with multiple outcomes.*

In this coin lemma we consider again the conjunction of several coin events that involve disjoint occurrences of actions. However we allow more freedom. First of all an action is paired with more than one set of states, thus allowing the observation of more than one outcome; second, we allow for multiple joint observations.

As an example, the coin lemma says that if n processes run in parallel and each one of them can flip a coin, then the probability that at least half of the processes either do not flip a coin or flip head is at least 1/2. Similarly, if each process can roll a dice, then the probability that if process 1 rolls 1 then the other processes do not roll a number different from 1 is at least $(1/6)^n + 5/6$, which is essentially the probability of rolling n dices and that either all processes give 1 or process 1 does not give 1.

6. *A generalized coin lemma.*

   In this coin lemma we generalize the idea of 5, but this time we do not use actions to identify the random draws of interest. The reader is referred to Section 6.4.2 for further details.

6.2.1 First Occurrence of an Action

Let M be a probabilistic automaton, and let (a, U) be a pair consisting of an action of M and a set of states of M. Let $\mathit{FIRST}(a, U)$ be a function that applied to a probabilistic execution fragment H of M returns the set of executions α of Ω_H such that either a does not occur in $\alpha \triangleright q_0^H$, or a occurs in $\alpha \triangleright q_0^H$ and the state reached after the first occurrence of a is a state of U.

It is simple to check that $\mathit{FIRST}(a, U)$ is an event schema since, for each probabilistic execution fragment H of M, the complement of $\mathit{FIRST}(a, U)(H)$ is the set of executions α of Ω_H such that action a occurs in $\alpha \triangleright q_0^H$, and the state reached after the first occurrence of a is not a state of U. This set is expressible as a union of cones, and thus it is an event.

The event schema $\mathit{FIRST}(a, U)$ identifies the first random draw associated with action a that occurs in a probabilistic execution fragment H, and requires the outcome of the random draw to be in a specific range, namely in U. The intuition behind the use of such a coin event, is that a system performs well if the outcome of the first random draw involving a is in U. From the definition of $\mathit{FIRST}(a, U)$, we assume also that the system performs well whenever a does not occur at all. Thus, if an adversary has the possibility not to schedule a, then it has a better chance to degrade the performance of a system by scheduling a.

The following lemma provides a lower bound to the probability of $\mathit{FIRST}(a, U)$. Informally, it states that if whenever there is a transition of M that involves action a the occurrence of a implies that a state of U is reached with probability at least p, then p is a lower bound on the probability of $\mathit{FIRST}(a, U)$.

Lemma 6.2.1 *Let M be a probabilistic automaton, and let (a, U) be a pair consisting of an action of M and a set of states of M. Let p be a real number between 0 and 1 such that for each transition $(s, \mathcal{P})$ of M where $P[a] > 0$, $P[U|a] \geq p$. Then, for each probabilistic execution fragment H of M, $P_H[\mathit{FIRST}(a, U)(H)] \geq p$.*

Proof. For convenience denote $\mathit{FIRST}(a, U)(H)$ by E, and for each state q of H, denote by $\Omega(q, \overline{U})$ the set $\{(a, q') \in \Omega_q^H \mid \mathit{lstate}(q') \notin U\}$. Let Θ be the set of states q of H such that action a does not occur in $q \triangleright q_0^H$, and $P_q^H[a] > 0$. Then,

$$P_H[\overline{E}] = \sum_{q\in\Theta} \sum_{(a,q')\in\Omega(q,\overline{U})} P_H[C_q] P_q^H[(a,q')]. \tag{6.1}$$

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

$$P_H[\overline{E}] = \sum_{q\in\Theta} P_H[C_q] P_q^H[a] \left(\sum_{(a,q')\in\Omega(q,\overline{U})} P_q^H[(a,q')|a] \right). \tag{6.2}$$

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

$$\sum_{(a,q')\in\Omega(q,\overline{U})} P_q^H[(a,q')|a] = P_{tr}[\overline{U}|a] = \frac{P_{tr}[\overline{U}\cap a]}{P_{tr}[a]} = \frac{\sum_i p_i P_{tr_i}[\overline{U}\cap a]}{\sum_i p_i P_{tr_i}[a]}. \tag{6.3}$$

By multiplying and dividing each i^{th} summand of the enumerator by $P_{tr_i}[a]$, using the hypothesis of the lemma, i.e., for each i $P_{tr_i}[\overline{U}\cap a] \leq (1-p)$, and simplifying algebraically, from (6.3) we obtain

$$\sum_{(a,q')\in\Omega(q,\overline{U})} P_q^H[(a,q')|a] \leq (1-p). \tag{6.4}$$

By using (6.4) in (6.2) we obtain

$$P_H[\overline{E}] \leq (1-p) \left(\sum_{q\in\Theta} P_H[C_q] P_q^H[a] \right). \tag{6.5}$$

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

$$P_H[\overline{E}] \leq (1-p). \tag{6.6}$$

This completes the proof. ■

6.2.2 First Occurrence of an Action among Many

The event schema $FIRST(a,U)$ can be generalized to account for the first action that occurs among several possible ones. Let M be a probabilistic automaton, and let $(a_1,U_1),\ldots,(a_n,U_n)$ be pairs consisting of an action of M and a set of states of M such that the actions a_i are all distinct. Then define $FIRST((a_1,U_1),\ldots,(a_n,U_n))$ to be the function that applied to a probabilistic execution fragment H of M returns the set of executions α of Ω_H such that either none of the a_i's occurs in $\alpha \triangleright q_0^H$, or some of the a_i's occur in $\alpha \triangleright q_0^H$, and if a_i is the first of those actions that occurs, then the state reached after the first occurrence of a_i is a state of U_i.

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

Lemma 6.2.1 extends to this case.

Lemma 6.2.2 *Let M be a probabilistic automaton, and let $(a_1, U_1), \ldots, (a_n, U_n)$ be pairs consisting of an action of M and a set of states of M such that the actions a_i are all distinct. Let $\{p_i\}_{i=1,\ldots,n}$ be a collection of real numbers between 0 and 1 such that for each i, $1 \leq i \leq n$, and each transition $(s, \mathcal{P})$ of M where $P[a_i] > 0$, $P[U|a_i] \geq p_i$. Then, for each probabilistic execution fragment H of M, $P_H[\mathit{FIRST}((a_1, U_1), \ldots, (a_n, U_n))(H)] \geq \mathit{min}(p_1, \ldots, p_n)$.*

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

$$P_q^H[\Omega(q, \overline{E})|V] \leq (1 - p). \tag{6.7}$$

To prove (6.7), let, for each $i = 1, \ldots, n$, $\Omega(q, a_i, \overline{U}_i)$ denote the set $\{(a_i, q') \in \Omega_q^H \mid \mathit{lstate}(q') \notin U_i\}$. Then,

$$P_q^H[\Omega(q, \overline{E})|V] = \sum_{i \in \{1,\ldots,n\}} P_q^H[\Omega(q, a_i, \overline{U}_i)|V]. \tag{6.8}$$

By using conditional probabilities, Equation (6.8) can be rewritten into

$$P_q^H[\Omega(q, \overline{E})|V] = \sum_{i \in \{1,\ldots,n\}} P_q^H[a_i|V] P_q^H[\Omega(q, a_i, \overline{U}_i)|a_i]. \tag{6.9}$$

Following the same argument as in the proof of Lemma 6.2.1, for each i, $P_q^H[\Omega(q, a_i, \overline{U}_i)|a_i] \leq (1 - p)$; moreover, $\sum_i P_q^H[a_i|V] = 1$. Thus, (6.7) follows directly.

The rest of the proof follows te lines of the proof of Lemma 6.2.1. Let Θ be the set of states q of H such that no action of V occurs in $q \triangleright q_0^H$, and $P_q^H[V] > 0$. Then,

$$P_H[\overline{E}] = \sum_{q \in \Theta} \sum_{(a,q') \in \Omega(q,\overline{E})} P_H[C_q] P_q^H[(a, q')]. \tag{6.10}$$

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

$$P_H[\overline{E}] = \sum_{q \in \Theta} P_H[C_q] P_q^H[V] \left(\sum_{(a,q') \in \Omega(q,\overline{E})} P_q^H[(a, q')|V] \right). \tag{6.11}$$

The subexpression $\sum_{(a,q') \in \Omega(q,\overline{E})} P_q^H[(a, q')|V]$ is $P_q^H[\Omega(q, \overline{E})|V]$, which is less than or equal to $(1 - p)$ from (6.7). Thus,

$$P_H[\overline{E}] \leq (1 - p) \left(\sum_{q \in \Theta} P_H[C_q] P_q^H[V] \right). \tag{6.12}$$

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

$$P_H[\overline{E}] \leq (1 - p). \tag{6.13}$$

This completes the proof. ■

6.2.3 I-th Occurrence of an Action among Many

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

Let M be a probabilistic automaton, and let $(a_1, U_1), \ldots, (a_n, U_n)$ be pairs consisting of an action of M and a set of states of M such that the actions a_i are all distinct. Then define $OCC(i, (a_1, U_1), \ldots, (a_n, U_n))$ to be the function that applied to a probabilistic execution fragment H of M returns the set of executions α of Ω_H such that either there are less than i occurrences of actions from $\{a_1, \ldots, a_n\}$ in $\alpha \triangleright q_0^H$, or there are at least i occurrences of actions from $\{a_1, \ldots, a_n\}$, and, if a_j is the action that occurs as the i^{th} one, then the state reached after its occurrence is a state of U_i.

Since in the proof of Lemma 6.2.2 we never use the fact that it is the first occurrence of an action that is considered, Lemma 6.2.2 carries over to the i^{th} occurrence trivially.

Lemma 6.2.3 *Let M be a probabilistic automaton, and let $(a_1, U_1), \ldots, (a_n, U_n)$ be pairs consisting of an action of M and a set of states of M such that the actions a_i are all distinct. Let $\{p_j\}_{j=1,\ldots,n}$ be a collection of real numbers between 0 and 1 such that for each $j \in \{1, \ldots, n\}$ and each transition $(s, \mathcal{P})$ of M where $P[a_j] > 0$, $P[U|a_j] \geq p_j$. Then, for each probabilistic execution fragment H of M, $P_H[OCC(i, (a_1, U_1), \ldots, (a_n, U_n))(H)] \geq min(p_1, \ldots, p_n)$.* ■

6.2.4 Conjunction of Separate Coin Events

In this section we study what happens if we consider several events of the kind OCC together. In order to simplify the notation, we consider only event schemas of the kind $OCC(i, (a, U))$ since, as we have seen in the proof of Lemma 6.2.2, the case with multiple actions can be reduced to the case with a single action.

The lemma that we prove states that if we consider several separate coin events, i.e., coin events that involve different random draws, each one with its own lower bound, then the lower bound of their conjunction is the product of the lower bounds. In other words, an adversary can introduce dependencies by increasing the probability of the conjunction of events, but it can never decrease the probability below the value that we would get by considering all the events to be independent.

Lemma 6.2.4 *Let M be a probabilistic automaton, and let $(k_1, a_1, U_1), \ldots, (k_n, a_n, U_n)$ be a collection of triplets consisting of a natural number, an action of M and a set of states of M, such that the pairs (k_i, a_i) are all distinct. Let $\{p_j\}_{j=1,\ldots,n}$ be a collection of real numbers between 0 and 1 such that for each $j \in \{1, \ldots, n\}$ and each transition $(s, \mathcal{P})$ of M where $P[a_j] > 0$, $P[U|a_j] \geq p_j$. Then, for each probabilistic execution fragment H of M, $P_H[OCC(k_1, (a_1, U_1))(H) \cap \cdots \cap OCC(k_n, (a_n, U_n))(H)] \geq p_1 \cdots p_n$.*

Proof. For each $I \subseteq \{1, \ldots, n\}$, denote a generic event schema $\cap_{i \in I} OCC(k_i, (a_i, U_i))$ by e_I. For each $i = 1, \ldots, n$ and each state q of H, denote by $\Omega(q, i, U_i)$ the set $\{(a_i, q') \in \Omega_q^H \mid lstate(q') \in U_i\}$ of pairs where a_i occurs and U_i is reached, and denote by $\Omega(q, i, \overline{U}_i)$ the set $\{(a_i, q') \in \Omega_q^H \mid lstate(q') \notin U_i\}$ of pairs where a_i occurs and U_i is not reached. For each action

a and each state q of H, let $a(q)$ denote the number of occurrences of action a in $q \triangleright q_0^H$. For each $i = 1, \ldots, n$, let Θ_i be the set of states q of H such that each action $a_j, 1 \leq j \leq n$ occurs less than k_j times in $q \triangleright q_0^H$, action a_i occurs $k_i - 1$ times in $q \triangleright q_0^H$, and $P_q^H[a_i] > 0$. For each $i = 1, \ldots, n$ and each state q of H such that $a_i(q) < k_i$, let $OCC(k_i, (a_i, U_i)) \triangleright q$ denote the event schema $OCC(k_i - a_i(q), (a_i, U_i))$. Finally, for each $I \subseteq \{1, \ldots, n\}$ and each suitable state q of H, let $e_I \triangleright q$ denote the event schema $\cap_{i \in I} OCC(k_i, (a_i, U_i)) \triangleright q$.

We prove the lemma by induction on n. If $n = 1$, then the result follows directly from Lemma 6.2.1. Otherwise,

$$P_H[\overline{e_{1,\ldots,n}(H)}] = \sum_{i \in \{1,\ldots,n\}} \sum_{q \in \Theta_i} P_H[C_q] \left(\left(\sum_{(a_i,q') \in \Omega(q,i,\overline{U}_i)} P_q^H[(a_i,q')] \right) \right.$$
$$\left. + \left(\sum_{(a_i,q') \in \Omega(q,i,U_i)} P_q^H[(a_i,q')] P_{H \triangleright q'}[\overline{e_{\{1,\ldots,i-1,i+1,\ldots,n\}} \triangleright q'(H \triangleright q')}] \right) \right). \qquad (6.14)$$

The first summand of Expression (6.14) expresses the probability that action a_i occurs from q and leads to a state not in U_i; the second summand expresses the probability that a_i occurs, leads to a state of U_i, and from the reached state something happen so that the resulting execution is not in $e_{1,\ldots,n}(H)$. From induction, and by using conditional probabilities, we obtain

$$P_H[\overline{e_{1,\ldots,n}(H)}] \leq \sum_{i \in \{1,\ldots,n\}} \sum_{q \in \Theta_i} P_H[C_q] P_q^H[a_i] \left(\left(\sum_{(a_i,q') \in \Omega(q,i,\overline{U}_i)} P_q^H[(a_i,q')|a_i] \right) \right.$$
$$\left. + \left(\sum_{(a_i,q') \in \Omega(q,i,U_i)} P_q^H[(a_i,q')|a_i])(1 - p_1 \cdots p_{i-1} p_{i+1} \cdots p_n) \right) \right). \qquad (6.15)$$

Let, for each i and each q, $p_{i,q} = P_q^H[\Omega(q,i,U_i)|a_i]$. Then, (6.15) becomes

$$P_H[\overline{e_{1,\ldots,n}(H)}]$$
$$\leq \sum_{i \in \{1,\ldots,n\}} \sum_{q \in \Theta_i} P_H[C_q] P_q^H[a_i]((1 - p_{i,q}) + (1 - p_1 \cdots p_{i-1} p_{i+1} \cdots p_n) p_{i,q}), \qquad (6.16)$$

which becomes

$$P_H[\overline{e_{1,\ldots,n}(H)}] \leq \sum_{i \in \{1,\ldots,n\}} \sum_{q \in \Theta_i} P_H[C_q] P_q^H[a_i](1 - p_1 \cdots p_{i-1} p_{i,q} p_{i+1} \cdots p_n) \qquad (6.17)$$

after simple algebraic simplifications. Using the same argument as in the proof of Lemma 6.2.1, for each i and each q, $p_{i,q} \geq p_i$. Thus,

$$P_H[\overline{e_{1,\ldots,n}(H)}] \leq \sum_{i \in \{1,\ldots,n\}} \sum_{q \in \Theta_i} P_H[C_q] P_q^H[a_i](1 - p_1 \cdots p_n). \qquad (6.18)$$

Finally, observe that $\sum_{i \in \{1,\ldots,n\}} \sum_{q \in \Theta_i} P_H[C_q] P_q^H[a_i]$ is the probability that for some i action a_i occurs at least k_i times. Thus,

$$P_H[\overline{e_{1,\ldots,n}(H)}] \leq (1 - p_1 \cdots p_n). \qquad (6.19)$$

This completes the proof. ■

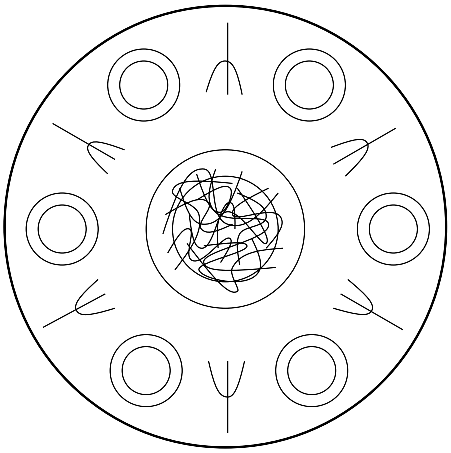


Figure 6-1: The Dining Philosopher problem with 6 philosophers.

6.3 Example: Randomized Dining Philosophers

In this section we apply the methodology presented so far to prove the correctness of the Randomized Dining Philosophers algorithm of Lehmann and Rabin [LR81]. The proof is structured in two levels. The high level proof consists of a collection of progress statements that are concatenated together; the low level proof consists of the proofs of the statements of the high level proof. The low level proof is based on the coin lemmas.

6.3.1 The Problem

There are n philosophers sat at a round table. Each philosopher has a plate in from of him, a fork on its left, and a fork on its right. The left fork is shared with his left neighbor philosopher, and the right fork is shared with his right neighbor philosopher. At the center of the table there is a bowl full of spaghetti. Figure 6-1 illustrates the situation for $n = 6$. Each philosopher goes repeatedly through phases where he is thinking and where he is eating. However, each philosopher needs both of its forks in order to eat. The problem is the following:

> "*What procedure should each philosopher follow to get his forks and to put them down in order to make sure that every philosopher that is hungry will eventually be able to eat?*"

A simpler problem is the following.

> "*What procedure should each philosopher follow to get his forks and to put them down in order to make sure that whenever somebody is hungry somebody will eventually be able to eat?*"

The second problem is simpler than the first problem since it allows for some philosopher to starve. It is known from [LR81] that there is no symmetric solution even for the simple dining philosophers problem, i.e., there is no deterministic solution for the dining philosophers problem where each philosopher follows exactly the same protocol; some mechanism to break the symmetry is necessary. In the algorithm of Lehmann and Rabin each philosopher follows exactly the same protocol and randomness is used to break the symmetry.

Shared variables: $\text{Res}_j \in \{\texttt{free}, \texttt{taken}\}$, $j = 1, \ldots, n$, initially free.

Local variables: $u_i \in \{\texttt{left}, \texttt{right}\}$, $i = 1, \ldots, n$

Code for process i**:**

```
0.    try                                          ** beginning of Trying Section **
1.    < $u_i \leftarrow random$>        ** choose left or right with equal probability **
2.    < if $\text{Res}_{(i,u_i)}$ = free then
            $\text{Res}_{(i,u_i)}$ := taken                  ** pick up first resource **
        else goto 2. >
3.     < if $\text{Res}_{(i,opp(u_i))}$ = free then
            $\text{Res}_{(i,opp(u_i))}$ := taken;           ** pick up second resource **
            goto 5. >
4.     < $\text{Res}_{(i,u_i)}$ := free; goto 1.>             ** put down first resource **
5.     crit                                              ** end of Trying Section **
       ** Critical Section **
6.     exit                                        ** beginning of Exit Section **
7.     < $u_i \leftarrow$ left or right                    ** nondeterministic choice **
          $\text{Res}_{(i,opp(u_i))}$ := free >            ** put down first resources **
8.     < $\text{Res}_{(i,u_i)}$ := free >                 ** put down second resources **
9.     rem                                                ** end of Exit Section **
       ** Remainder Section **
```

Figure 6-2: The Lehmann-Rabin algorithm. The operations between angular brackets are performed atomically.

6.3.2 The Algorithm

Each hungry philosopher proceeds according to the following protocol.

1. Flip a fair coin to choose between the left and the right fork.
2. Wait for the chosen fork to become free and get it.
3. Try to get the second fork:
    if it is free, then get it;
    if it is taken, then put down the first fork and go to 1.
4. Eat.

Each philosopher that has terminated to eat puts down his forks one at a time. The intuition behind the use of randomness is that the actual protocol used by each philosopher is determined by an infinite sequence of random coin flips. Thus, with probability 1 each philosopher follows a different protocol.

Figure 6-2 gives a more precise representation of the protocol, using a terminology that is closer to computer science; thus, a philosopher is called a process, and a fork is called a resource. A philosopher who is thinking is said to be in its *reminder* region; a philosopher

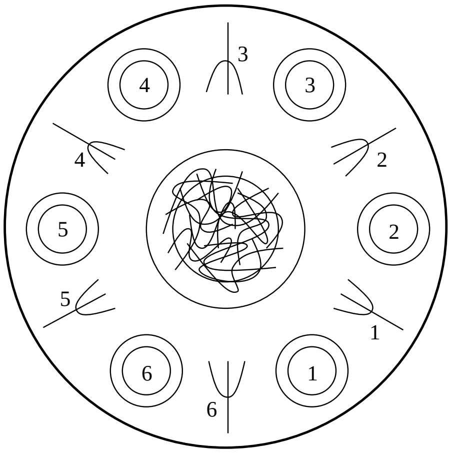


Figure 6-3: Numbering processes and resources in the Dining Philosophers problem.

who is eating is said to be in its *critical* region; a philosopher who is trying to get its forks is said to be in its *trying* region; and a philosopher who is putting down its forks is said to be in its *exit* region. The n resources (forks) are represented by n shared variables $\mathrm{Res}_1, \ldots, \mathrm{Res}_n$, each of which can assume values in $\{\mathtt{free}, \mathtt{taken}\}$. Each process (philosopher) i ignores its own name and the names of its adjacent resources. However, each process i is able to refer to its adjacent resources by relative names: $\mathrm{Res}_{(i,\mathtt{left})}$ is the resource located to the left, and $\mathrm{Res}_{(i,\mathtt{right})}$ is the resource to the right of i. Each process i has a private variable u_i, whose value is in $\{\mathtt{left}, \mathtt{right}\}$, which is used either to keep track of the resource that process i currently holds, or, if no resource is held, to keep track of the resource that process i is going to take next. For notational convenience we define an operator *opp* that complements the value of its argument, i.e., $opp(\mathtt{right}) = \mathtt{left}$ and $opp(\mathtt{left}) = \mathtt{right}$.

We now define a probabilistic automaton M that represents the evolution of n philosophers. We assume that process $i+1$ is on the right of process i and that resource Res_i is between processes i and $i+1$ (see Figure 6-3). We also identify labels modulo n so that, for instance, process $n+1$ coincides with process 1.

A state s of M is a tuple $(X_1, \ldots, X_n, \mathrm{Res}_1, \ldots, \mathrm{Res}_n)$ containing the local state X_i of each process i, and the value of each resource Res_i. Each local state X_i is a pair (pc_i, u_i) consisting of a program counter pc_i and the local variable u_i. The program counter of each process keeps track of the current instruction in the code of Figure 6-2. Rather than representing the value of the program counter with a number, we use a more suggestive notation which is explained in Table 6.1. Also, the execution of each instruction is represented by an action. Actions $\mathtt{try}_i$, $\mathtt{crit}_i$, $\mathtt{rem}_i$, $\mathtt{exit}_i$ are external; all the other actions are internal.

The start state of M assigns the value $\mathtt{free}$ to all the shared variables Res_i, the value R to each program counter pc_i, and an arbitrary value to each variable u_i. The transition relation of M is derived directly from Figure 6-2. For example, for each state where $pc_i = F$ there is an internal transition labeled with $\mathtt{flip}_i$ that changes pc_i into W and assigns $\mathtt{left}$ to u_i with probability 1/2 and $\mathtt{right}$ to u_i with probability 1/2; from each state where $X_i = (W, \mathtt{left})$ there is a transition labeled with $\mathtt{wait}_i$ that does not change the state if $\mathrm{Res}_{(i,\mathtt{left})} = \mathtt{taken}$, and changes pc_i into S and $\mathrm{Res}_{(i,\mathtt{left})}$ into $\mathtt{taken}$ if $\mathrm{Res}_{(i,\mathtt{left})} = \mathtt{free}$; for each state where

Nr.	pc_i	Action	Informal meaning
0	R	$\texttt{try}_i$	**R**eminder region
1	F	$\texttt{flip}_i$	Ready to **F**lip
2	W	$\texttt{wait}_i$	**W**aiting for first resource
3	S	$\texttt{second}_i$	Checking for **S**econd resource
4	D	$\texttt{drop}_i$	**D**ropping first resource
5	P	$\texttt{crit}_i$	**P**re-critical region
6	C	$\texttt{exit}_i$	**C**ritical region
7	E_F	$\texttt{dropf}_i$	**E**xit: drop **F**irst resource
8	E_S	$\texttt{drops}_i$	**E**xit: drop **S**econd resource
9	E_R	$\texttt{rem}_i$	**E**xit: move to **R**eminder region

Table 6.1: Program counter and action names for the Lehmann-Rabin algorithm.

$pc_i = E_F$ there are two transitions labeled with action $\texttt{dropf}_i$: one transition sets u_i to $\texttt{right}$ and makes $\text{Res}_{(i,\texttt{left})}$ free, and the other transition sets u_i to $\texttt{left}$ makes $\text{Res}_{(i,\texttt{right})}$ free. The two separate transitions correspond to a nondeterministic choice that is left to the adversary.

The value of each pair X_i can be represented concisely by the value of pc_i and an arrow (to the left or to the right) which describes the value of u_i. Thus, informally, a process i is in state $\underset{\rightarrow}{S}$ or $\underset{\rightarrow}{D}$ (resp. $\underset{\leftarrow}{S}$ or $\underset{\leftarrow}{D}$) when i is in state S or D while holding its right (resp. left) resource; process i is in state $\underset{\rightarrow}{W}$ (resp. $\underset{\leftarrow}{W}$) when i is waiting for its right (resp. left) resource to become free; process i is in state $\underset{\rightarrow}{E_S}$ (resp. $\underset{\leftarrow}{E_S}$) when i is in its exit region and it is still holding its right (resp. left) resource. Sometimes we are interested in sets of pairs; for example, whenever $pc_i = F$ the value of u_i is irrelevant. With the simple value of pc_i we denote the set of the two pairs $\{(pc_i, \texttt{left}), (pc_i, \texttt{right})\}$. Finally, with the symbol # we denote any pair where $pc_i \in \{W, S, D\}$. The arrow notation is used as before.

For each state $s = (X_1, \ldots, X_n, \text{Res}_1, \ldots, \text{Res}_n)$ of M we denote X_i by $X_i(s)$ and Res_i by $\text{Res}_i(s)$. Also, for any set St of states of a process i, we denote by $X_i \in St$, or alternatively $X_i = St$ the set of states s of M such that $X_i(s) \in St$. Sometimes we abuse notation in the sense that we write expressions like $X_i \in \{F, D\}$ with the meaning $X_i \in F \cup D$. Finally, we write $X_i = E$ for $X_i = \{E_F, E_S, E_R\}$, and we write $X_i = T$ for $X_i \in \{F, W, S, D, P\}$.

6.3.3 The High Level Proof

In this section we give the high level proof that the algorithm of Lehmann and Rabin guarantees progress, i.e., that from every state where some process is in its trying region, some process enters eventually its critical region with probability 1. We assume that each process that is ready to perform a transition is allowed eventually to do so: process i is ready to perform a transition whenever it enables an action different from $\texttt{try}_i$ or $\texttt{exit}_i$. Actions $\texttt{try}_i$ and $\texttt{exit}_i$ are under the control of the user (a philosopher decides whether to eat or think), and hence, by assumption, under the control of the adversary.

Formally, consider the probabilistic automaton M of Section 6.3.2. Define an extended execution α of M to be *fair* iff for each process i either α is finite and its last state enables

$\mathtt{try}_i$ or $\mathtt{exit}_i$, or α is infinite and either actions of process i occur infinitely many times in α or $\alpha = \alpha_1 \frown \alpha_2$ and all the states of α_2 enable either $\mathtt{try}_i$ or $\mathtt{exit}_i$. Define *Fairadvs* to be the set of adversaries $\mathcal{A}$ for M such that, for every finite execution fragment α of M the elements of $\Omega_{prexec(M,\mathcal{A},\alpha)}$ are extended fair execution fragments of M. Then *Fairadvs* is finite-history-insensitive: if $\mathcal{A}$ is an adversary of *Fairadvs* and q is a finite execution fragment of M, then it is easy to verify that the adversary $\mathcal{A}_q$ such that

$$\mathcal{A}_q(\alpha) = \begin{cases} \mathcal{A}(\alpha \triangleright q) & \text{if } q \leq \alpha \\ \mathcal{A}(\alpha) & \text{otherwise} \end{cases}$$

is an adversary of *Fairadvs*. Let $rstates(M)$ denote the set of reachable states of M. Let

$$\mathcal{T} \ \triangleq\ \{s \in rstates(M) \mid \exists_i X_i(s) \in \{T\}\}$$

denote the sets of reachable states of M where some process is in its trying region, and let

$$\mathcal{C} \ \triangleq\ \{s \in rstates(M) \mid \exists_i X_i(s) = C\}$$

denote the sets of reachable states of M where some process is in its critical region. We first show that

$$\mathcal{T} \underset{1/8}{\longrightarrow}_{Fairadvs} \mathcal{C}, \tag{6.20}$$

i.e., that, starting from any reachable state where some process is in its trying region, for all the adversaries of *Fairadvs*, some process enters its critical region eventually with probability at least 1/8. Note that (6.20) is satisfied trivially if some process is initially in its critical region.

Our proof is divided into several phases, each one concerned with the property of making some partial progress toward $\mathcal{C}$. The sets of states associated with the different phases are expressed in terms of $\mathcal{T}, \mathcal{RT}, \mathcal{F}, \mathcal{G}, \mathcal{P}$, and $\mathcal{C}$. Here,

$$\mathcal{RT} \ \triangleq\ \{s \in \mathcal{T} \mid \forall_i X_i(s) \in \{E_R, R, T\}\}$$

is the set of states where at least one process is in its trying region and where no process is in its critical region or holds resources while being in its exit region.

$$\mathcal{F} \ \triangleq\ \{s \in \mathcal{RT} \mid \exists_i X_i(s) = F\}$$

is the set of states of $\mathcal{RT}$ where some process is ready to flip a coin.

$$\mathcal{P} \ \triangleq\ \{s \in rstates(M) \mid \exists_i X_i(s) = P\}$$

is the sets of reachable states of M where some process is in its pre-critical region, i.e., where some process is ready to enter its critical region. The set $\mathcal{G}$ is the most important for the analysis. To motivate the definition, we define the following notions. We say that a process i is *committed* if $X_i \in \{W, S\}$, and that a process *i potentially controls* Res_i (resp. Res_{i-1}) if $X_i \in \{\underrightarrow{W}, \underrightarrow{S}, \underrightarrow{D}\}$ (resp. $X_i \in \{\underleftarrow{W}, \underleftarrow{S}, \underleftarrow{D}\}$). Informally said, a state in $\mathcal{RT}$ is in $\mathcal{G}$ if and only if there is a committed process whose second resource is not potentially controlled by another process. Such a process is called a *good* process. Formally,

$$\begin{aligned} \mathcal{G} \ \triangleq\ \{s \in \mathcal{RT} \mid \exists_i \\ & X_i(s) \in \{\underleftarrow{W}, \underleftarrow{S}\} \text{ and } X_{i+1}(s) \in \{E_R, R, F, \underrightarrow{\#}\}, \text{ or} \\ & X_i(s) \in \{\underrightarrow{W}, \underrightarrow{S}\} \text{ and } X_{i-1}(s) \in \{E_R, R, F, \underleftarrow{\#}\}\} \end{aligned}$$

Reaching a state of $\mathcal{G}$ is a substantial progress toward reaching a state of $\mathcal{C}$. Somehow, a good state is a place where the symmetry is broken. The progress statements of the proof are the following.

$$\begin{array}{ll}
\mathcal{T} \xrightarrow[1]{} \mathcal{RT} \cup \mathcal{C} & \text{(Proposition 6.3.3)}, \\
\mathcal{RT} \xrightarrow[1]{} \mathcal{F} \cup \mathcal{G} \cup \mathcal{P} & \text{(Proposition 6.3.16)}, \\
\mathcal{F} \xrightarrow[1/2]{} \mathcal{G} \cup \mathcal{P} & \text{(Proposition 6.3.15)}, \\
\mathcal{G} \xrightarrow[1/4]{} \mathcal{P} & \text{(Proposition 6.3.12)}, \\
\mathcal{P} \xrightarrow[1]{} \mathcal{C} & \text{(Proposition 6.3.1)}.
\end{array}$$

The first statement says that eventually every process in its exit region relinquishes its resources. In this way we avoid to deal with resources held by processes who do not want to enter the critical region. The second statement says that eventually either a good state is reached, or a place where some process is ready to flip its coin is reached. The flipping points are potential points where the symmetry is broken, and thus reaching a flipping point means progress. The third statement says that from a flipping point there is probability 1/2 to reach a good state. Finally, the fourth statement says that from a good state there is probability 1/4 to be ready to enter the critical region. By combining the statements above by means of Proposition 5.5.3 and Theorem 5.5.2 we obtain

$$\mathcal{T} \xrightarrow[1/8]{} \mathcal{C}, \tag{6.21}$$

which is the property that was to be proven. Observe that once some process is in the trying region there is always some process in the trying region until some process reaches the critical region. Formally, M satisfies $\mathcal{T}$ *Unless* $\mathcal{C}$. Thus, Proposition 5.5.6 applies, leading to

$$\mathcal{T} \xrightarrow[1]{} \mathcal{C}. \tag{6.22}$$

6.3.4 The Low Level Proof

In this section we prove the five progress statements used in Section 6.3.3. The proofs are detailed operational arguments. The main point to observe is that randomness is handled exclusively by the coin lemmas, and thus, any technique for the verification of ordinary automata could be applied as well.

For the sake of clarity, we do not prove the relations in the order they were presented. Throughout the proof we abuse notation by writing expressions of the kind $\mathit{FIRST}(\mathtt{flip}_i, \mathtt{left})$ for the event schema $\mathit{FIRST}(\mathtt{flip}_i, \{s \in \mathit{states}(M) \mid X_i(s) = \underleftarrow{W}\})$. We write also sentences of the form "If $\mathit{FIRST}(\mathtt{flip}_i, \mathtt{left})$ then ϕ" meaning that for each valid probabilistic execution fragment H, each element of $\mathit{FIRST}(\mathtt{flip}_i, \mathtt{left})(H)$ satisfies ϕ.

Proposition 6.3.1 *If some process is in P, then some process enters C, i.e.,*

$$\mathcal{P} \xrightarrow[1]{} \mathcal{C}.$$

Proof. Let i be the process in P. Then, from the definition of *Fairadvs*, process i is scheduled eventually, and enters C. ■

Lemma 6.3.2 *If some process is in its Exit region, then it will eventually enter R.*

Proof. The process needs to perform two transitions to relinquish its two resources, and then one transition to send a `rem` message to the user. Every adversary of *Fairadvs* guarantees that those three transitions are performed eventually. ■

Proposition 6.3.3 $\mathcal{T} \longrightarrow \mathcal{RT} \cup \mathcal{C}$.

Proof. From Lemma 6.3.2, every process that begins in E_F or E_S relinquishes its resources. If no process begins in C or enters C in the meantime, then the state reached at this point is a state of $\mathcal{RT}$; otherwise, the starting state or the state reached when the first process enters C is a state of $\mathcal{C}$. ■

We now turn to the proof of $\mathcal{G} \underset{1/4}{\longrightarrow} \mathcal{P}$. The following lemmas form a detailed cases analysis of the different situations that can arise in states of $\mathcal{G}$. Informally, each lemma shows that a specific coin event is a sub-event of the properties of reaching some other state. A preliminary lemma is an invariant of M, which guarantees that the resources are held by those processes who think to be holding them.

Lemma 6.3.4 *For each reachable state s of M and each i, $1 \le i \le n$, $Res_i =$ `taken` iff $X_i(s) \in \{\underrightarrow{S}, \underrightarrow{D}, P, C, E_F, \underrightarrow{E_S}\}$ or $X_{i+1}(s) \in \{\underleftarrow{S}, \underleftarrow{D}, P, C, E_F, \underleftarrow{E_S}\}$. Moreover, for each reachable state s of M and each i, $1 \le i \le n$, it is not the case that $X_i(s) \in \{\underrightarrow{S}, \underrightarrow{D}, P, C, E_F, \underrightarrow{E_S}\}$ and $X_{i+1}(s) \in \{\underleftarrow{S}, \underleftarrow{D}, P, C, E_F, \underleftarrow{E_S}\}$, i.e., only one process at a time can hold one resource.* ■

Proof. The proof of this lemma is a standard proof of invariants. Simply verify that the two properties are true for the start states of M and are preserved by each transition of M. ■

Lemma 6.3.5

1. *Let $X_{i-1} \in \{E_R, R, F\}$ and $X_i = \underleftarrow{W}$. If FIRST(`flip`$_{i-1}$, `left`), then, eventually, either $X_{i-1} = P$ or $X_i = S$.*

2. *Let $X_{i-1} = D$ and $X_i = \underleftarrow{W}$. If FIRST(`flip`$_{i-1}$, `left`), then, eventually, either $X_{i-1} = P$ or $X_i = S$.*

3. *Let $X_{i-1} = S$ and $X_i = \underleftarrow{W}$. If FIRST(`flip`$_{i-1}$, `left`), then, eventually, either $X_{i-1} = P$ or $X_i = S$.*

4. *Let $X_{i-1} = W$ and $X_i = \underleftarrow{W}$. If FIRST(`flip`$_{i-1}$, `left`), then, eventually, either $X_{i-1} = P$ or $X_i = S$.*

Proof. The four proofs start in the same way. Let s be a state of M satisfying the respective properties of items *1* or *2* or *3* or *4*. Let $\mathcal{A}$ be an adversary of *Fairadvs*, and let α be an execution of $\Omega_{prexec(M,\{s\},\mathcal{A})}$ where the result of the first coin flip of process $i-1$, if it occurs, is `left`.

1. By hypothesis and Lemma 6.3.4, $i-1$ does not hold any resource at the beginning of α and has to obtain Res_{i-2} (its left resource) before pursuing Res_{i-1}. From the definition of *Fairadvs*, i performs a transition eventually in α. If $i-1$ does not hold Res_{i-1} when i performs this transition, then i progresses into configuration S. If not, it must be the case that $i-1$ succeeded in getting it in the meanwhile. But, in this case, since $i-1$ flips `left`, Res_{i-1} was the second resource needed by $i-1$ and $i-1$ therefore entered P.

2. If $X_i = S$ eventually, then we are done. Otherwise, process $i-1$ performs a transition eventually. Let $\alpha = \alpha_1 {}^\frown \alpha_2$ such that the last transition of α_1 is the first transition taken by process $i-1$. Then $X_{i-1}(\mathit{fstate}(\alpha_2)) = F$ and $X_i(\mathit{fstate}(\alpha_2)) = \underleftarrow{W}$. Since process $i-1$ did not flip any coin during α_1, from the finite-history-insensitivity of *Fairadvs* and Item *1* we conclude.

3. If $X_i = S$ eventually, then we are done. Otherwise, process $i-1$ performs a transition eventually. Let $\alpha = \alpha_1 {}^\frown \alpha_2$ such that the last transition of α_1 is the first transition taken by process $i-1$. If $X_{i-1}(\mathit{fstate}(\alpha_2)) = P$ then we are also done. Otherwise it must be the case that $X_{i-1}(\mathit{fstate}(\alpha_2)) = D$ and $X_i(\mathit{fstate}(\alpha_2)) = \underleftarrow{W}$. Since process $i-1$ did not flip any coin during α_1, from the finite-history-insensitivity of *Fairadvs* and Item *2* we conclude.

4. If $X_i = S$ eventually, then we are done. Otherwise, process i checks its left resource eventually and fails, process $i-1$ gets its right resource before, and hence reaches at least state S. Let $\alpha = \alpha_1 {}^\frown \alpha_2$ where the last transition of α_1 is the first transition of α that leads process $i-1$ to state S. Then $X_{i-1}(\mathit{fstate}(\alpha_2)) = S$ and $X_i(\mathit{fstate}(\alpha_2)) = \underleftarrow{W}$. Since process $i-1$ did not flip any coin during α_1, from the finite-history-insensitivity of *Fairadvs* and Item *3* we conclude. ■

Lemma 6.3.6 *Assume that* $X_{i-1} \in \{E_R, R, T\}$ *and* $X_i = \underleftarrow{W}$. *If* $\mathit{FIRST}(\mathtt{flip}_{i-1}, \mathtt{left})$, *then, eventually, either* $X_{i-1} = P$ *or* $X_i = S$.

Proof. Follows directly from Lemma 6.3.5 after observing that $X_{i-1} \in \{E_R, R, T\}$ is equivalent to $X_{i-1} \in \{E_R, R, F, W, S, D, P\}$. ■

The next lemma is a useful tool for the proofs of Lemmas 6.3.8, 6.3.9, and 6.3.10.

Lemma 6.3.7 *Let* $X_i \in \{\underleftarrow{W}, \underleftarrow{S}\}$ *or* $X_i \in \{E_R, R, F, \underleftarrow{D}\}$ *with* $\mathit{FIRST}(\mathtt{flip}_i, \mathtt{left})$. *Furthermore, let* $X_{i+1} \in \{\underrightarrow{W}, \underrightarrow{S}\}$ *or* $X_{i+1} \in \{E_R, R, F, \underrightarrow{D}\}$ *with* $\mathit{FIRST}(\mathtt{flip}_{i+1}, \mathtt{right})$. *Then the first of the two processes* i *or* $i+1$ *testing its second resource enters* P *after having performed this test (if this time ever comes).*

Proof. By Lemma 6.3.4 Res_i is free. Moreover, Res_i is the second resource needed by both i and $i+1$. Whichever tests for it first gets it and enters P. ■

Lemma 6.3.8 *If* $X_i = \underleftarrow{S}$ *and* $X_{i+1} \in \{\underrightarrow{W}, \underrightarrow{S}\}$ *then, eventually, one of the two processes* i *or* $i+1$ *enters* P. *The same result holds if* $X_i \in \{\underleftarrow{W}, \underleftarrow{S}\}$ *and* $X_{i+1} = \underrightarrow{S}$.

Proof. Being in state S, process i tests its second resource eventually. An application of Lemma 6.3.7 finishes the proof. ■

Lemma 6.3.9 *Let* $X_i = \underset{\leftarrow}{S}$ *and* $X_{i+1} \in \{E_R, R, F, \underset{\rightarrow}{D}\}$*. If* $FIRST(\mathtt{flip}_{i+1}, \mathtt{right})$*, then, eventually, one of the two processes* i *or* $i+1$ *enters* P*. The same result holds if* $X_i \in \{E_R, R, F, D\}$*,* $X_{i+1} = \underset{\rightarrow}{S}$ *and* $FIRST(\mathtt{flip}_i, \mathtt{left})$*.*

Proof. Being in state S, process i tests its second resource eventually. An application of Lemma 6.3.7 finishes the proof. ■

Lemma 6.3.10 *Assume that* $X_{i-1} \in \{E_R, R, T\}$*,* $X_i = \underset{\leftarrow}{W}$*, and* $X_{i+1} \in \{E_R, R, F, \underset{\rightarrow}{W}, \underset{\rightarrow}{D}\}$*. If* $FIRST(\mathtt{flip}_{i-1}, \mathtt{left})$ *and* $FIRST(\mathtt{flip}_{i+1}, \mathtt{right})$*, then eventually one of the three processes* $i-1$*,* i *or* $i+1$ *enters* P*.*

Proof. Let s be a state of M such that $X_{i-1}(s) \in \{E_R, R, T\}$, $X_i(s) = \underset{\leftarrow}{W}$, and $X_{i+1}(s) \in \{E_R, R, F, \underset{\rightarrow}{W}, \underset{\rightarrow}{D}\}$. Let $\mathcal{A}$ be an adversary of *Fairadvs*, and let α be an extended execution of $\Omega_{prexec(M,\{s\},\mathcal{A})}$ where the result of the first coin flip of process $i-1$ is `left` and the result of the first coin flip of process $i+1$ is `right`. By Lemma 6.3.6, eventually either process $i-1$ reaches configuration P in α or process i reaches configuration $\underset{\leftarrow}{S}$ in α. If $i-1$ reaches configuration P, then we are done. If not, then let $\alpha = \alpha_1 \frown \alpha_2$ such that $lstate(\alpha_1)$ is the first state s' of α with $X_i(s') = \underset{\leftarrow}{S}$. If $i+1$ enters P before the end of α_1, then we are done. Otherwise, $X_{i+1}(fstate(\alpha_2))$ is either in $\{\underset{\rightarrow}{W}, \underset{\rightarrow}{S}\}$ or it is in $\{E_R, R, F, \underset{\rightarrow}{D}\}$ and process $i+1$ has not flipped any coin yet in α. From the finite-history-insensitivity of *Fairadvs* we can then apply Lemma 6.3.7: eventually process i tests its second resource and by Lemma 6.3.7 process i enters P if process $i+1$ did not check its second resource in the meantime. If process $i+1$ checks its second resource before process i does the same, then by Lemma 6.3.7 process $i+1$ enters P. ■

Lemma 6.3.11 *Assume that* $X_{i+2} \in \{E_R, R, T\}$*,* $X_{i+1} = \underset{\rightarrow}{W}$*, and* $X_i \in \{E_R, R, F, \underset{\leftarrow}{W}, \underset{\leftarrow}{D}\}$*. If* $FIRST(\mathtt{flip}_i, \mathtt{left})$ *and* $FIRST(\mathtt{flip}_{i+2}, \mathtt{right})$*, then eventually one of the three processes* i*,* $i+1$ *or* $i+2$*, enters* P*.*

Proof. The proof is analogous to the one of Lemma 6.3.10. This lemma is the symmetric case of Lemma 6.3.10. ■

Proposition 6.3.12 *Starting from a global configuration in* $\mathcal{G}$*, then, with probability at least* $1/4$*, some process enters* P *eventually. Equivalently:*

$$\mathcal{G} \xrightarrow[1/4]{} \mathcal{P}.$$

Proof. Lemmas 6.3.8 and 6.3.9 jointly treat the case where $X_i = \underset{\leftarrow}{S}$ and $X_{i+1} \in \{E_R, R, F, \underset{\rightarrow}{\#}\}$ and the symmetric case where $X_i \in \{E_R, R, F, \underset{\leftarrow}{\#}\}$ and $X_{i+1} = \underset{\rightarrow}{S}$; Lemmas 6.3.10 and 6.3.11 jointly treat the case where $X_i = \underset{\leftarrow}{W}$ and $X_{i+1} \in \{E_R, R, F, \underset{\rightarrow}{W}, \underset{\rightarrow}{D}\}$ and the symmetric case where $X_i \in \{E_R, R, F, \underset{\leftarrow}{W}, \underset{\leftarrow}{D}\}$ and $X_{i+1} = \underset{\rightarrow}{W}$.

Specifically, each lemma shows that a compound event of the kind $\mathit{FIRST}(\texttt{flip}_i, x)$ and $\mathit{FIRST}(\texttt{flip}_j, y)$ leads to $\mathcal{P}$. Each of the basic events $\mathit{FIRST}(\texttt{flip}_i, x)$ has probability at least $1/2$. From Lemma 6.2.4 each of the compound events has probability at least $1/4$. Thus the probability of reaching $\mathcal{P}$ eventually is at least $1/4$. ■

We now turn to $\mathcal{F} \underset{1/2}{\longrightarrow} \mathcal{G} \cup \mathcal{P}$. The proof is divided in two parts and constitute the global argument of the proof of progress, i.e., the argument that focuses on the whole system rather than on a couple of processes.

Lemma 6.3.13 *Start with a state s of $\mathcal{F}$. If there exists a process i for which $X_i(s) = F$ and $(X_{i-1}, X_{i+1}) \neq (\underset{\rightarrow}{\#}, \underset{\leftarrow}{\#})$, then, with probability at least $1/2$ a state of $\mathcal{G} \cup \mathcal{P}$ is reached eventually.*

Proof. If $s \in \mathcal{G} \cup \mathcal{P}$, then the result is trivial. Let s be a state of $\mathcal{F} - (\mathcal{G} \cup \mathcal{P})$ and let i be such that $X_i(s) = F$ and $(X_{i-1}, X_{i+1}) \neq (\underset{\rightarrow}{\#}, \underset{\leftarrow}{\#})$. Assume without loss of generality that $X_{i+1} \neq \underset{\leftarrow}{\#}$, i.e., $X_{i+1} \in \{E_R, R, F, \underset{\rightarrow}{\#}\}$. The case for $X_{i-1} \neq \underset{\rightarrow}{\#}$ is similar. Furthermore, we can assume that $X_{i+1} \in \{E_R, R, F, \underset{\rightarrow}{D}\}$ since if $X_{i+1} \in \{\underset{\rightarrow}{W}, \underset{\rightarrow}{S}\}$ then s is already in $\mathcal{G}$. We show that the event schema $\mathit{FIRST}((\texttt{flip}_i, \texttt{left}), (\texttt{flip}_{i+1}, \texttt{right}))$, which by Lemma 6.2.2 has probability at least $1/2$, leads eventually to a state of $\mathcal{G} \cup \mathcal{P}$. Let $\mathcal{A}$ be an adversary of *Fairadvs*, and let α be an extended execution of $\Omega_{prexec(M,\{s\},\mathcal{A})}$ where if process i flips before process $i+1$ then process i flips left, and if process $i+1$ flips before process i then process $i+1$ flips right.

Then, eventually, i performs one transition and reaches W. Let $j \in \{i, i+1\}$ be the first of i and $i+1$ that reaches W and let s_1 be the state reached after the first time process j reaches W. If some process reached P in the meantime, then we are done. Otherwise there are two cases to consider. If $j = i$, then, $\texttt{flip}_i$ yields $\texttt{left}$ and $X_i(s_1) = \underset{\leftarrow}{W}$ whereas X_{i+1} is (still) in $\{E_R, R, F, \underset{\rightarrow}{D}\}$. Therefore, $s_1 \in \mathcal{G}$. If $j = i+1$, then $\texttt{flip}_{i+1}$ yields $\texttt{right}$ and $X_{i+1}(s_1) = \underset{\rightarrow}{W}$ whereas $X_i(s_1)$ is (still) F. Therefore, $s_1 \in \mathcal{G}$. ■

Lemma 6.3.14 *Start with a state s of $\mathcal{F}$. If there exists a process i for which $X_i(s) = F$ and $(X_{i-1}(s), X_{i+1}(s)) = (\underset{\rightarrow}{\#}, \underset{\leftarrow}{\#})$. Then, with probability at least $1/2$, a state of $\mathcal{G} \cup \mathcal{P}$ is reached eventually.*

Proof. The hypothesis can be summarized into the form $(X_{i-1}(s), X_i(s), X_{i+1}(s)) = (\underset{\rightarrow}{\#}, F, \underset{\leftarrow}{\#})$. Since $i-1$ and $i+1$ point in different directions, by moving to the right of $i+1$ there is a process k pointing to the left such that process $k+1$ either points to the right or is in $\{E_R, R, F, P\}$, i.e., $X_k(s) \in \{\underset{\leftarrow}{W}, \underset{\leftarrow}{S}, \underset{\leftarrow}{D}\}$ and $X_{k+1}(s) \in \{E_R, R, F, \underset{\rightarrow}{W}, \underset{\rightarrow}{S}, \underset{\rightarrow}{D}, P\}$.

If $X_k(s) \in \{\underset{\leftarrow}{W}, \underset{\leftarrow}{S}\}$ and $X_{k+1}(s) \neq P$ then $s \in \mathcal{G}$ and we are done; if $X_{k+1}(s) = P$ then $s \in \mathcal{P}$ and we are done. Thus, we can restrict our attention to the case where $X_k(s) = \underset{\leftarrow}{D}$.

We show that $\mathit{FIRST}((\texttt{flip}_k, \texttt{left}), (\texttt{flip}_{k+1}, \texttt{right}))$, which by Lemma 6.2.2 has probability at least $1/2$, leads eventually to $\mathcal{G} \cup \mathcal{P}$. Let $\mathcal{A}$ be an adversary of *Fairadvs*, and let α be an extended execution of $\Omega_{prexec(M,\{s\},\mathcal{A})}$ where if process k flips before process $k+1$ then process k flips left, and if process $k+1$ flips before process k then process $k+1$ flips right.

Then, eventually, process k performs at least two transitions and hence goes to configuration W. Let $j \in \{k, k+1\}$ be the first of k and $k+1$ that reaches W and let s_1 be the state reached after the first time process j reaches W. If some process reached P in the meantime, then we are done. Otherwise, we distinguish two cases. If $j = k$, then, $\texttt{flip}_k$ yields $\texttt{left}$ and $X_k(s_1) = \underleftarrow{W}$ whereas X_{k+1} is (still) in $\{E_R, R, F, \underrightarrow{\#}\}$. Therefore, $s_1 \in \mathcal{G}$. If $j = k+1$, then $\texttt{flip}_{k+1}$ yields $\texttt{right}$ and $X_{k+1}(s_1) = \underrightarrow{W}$ whereas $X_k(s_1)$ is (still) in $\{\underleftarrow{D}, F\}$. Therefore, $s_1 \in \mathcal{G}$. ■

Proposition 6.3.15 *Start with a state s of $\mathcal{F}$. Then, with probability at least $1/2$, a state of $\mathcal{G} \cup \mathcal{P}$ is reached eventually. Equivalently:*

$$\mathcal{F} \xrightarrow[1/2]{} \mathcal{G} \cup \mathcal{P}.$$

Proof. The hypothesis of Lemmas 6.3.13 and 6.3.14 form a partition of $\mathcal{F}$. ■

Finally, we prove $\mathcal{RT} \xrightarrow[1]{} \mathcal{F} \cup \mathcal{G} \cup \mathcal{P}$.

Proposition 6.3.16 *Starting from a state s of $\mathcal{RT}$, then a state of $\mathcal{F} \cup \mathcal{G} \cup \mathcal{P}$ is reached eventually. Equivalently:*

$$\mathcal{RT} \xrightarrow[1]{} \mathcal{F} \cup \mathcal{G} \cup \mathcal{P}.$$

Proof. Let s be a state of $\mathcal{RT}$. If $s \in \mathcal{F} \cup \mathcal{G} \cup \mathcal{P}$, then we are trivially done. Suppose that $s \notin \mathcal{F} \cup \mathcal{G} \cup \mathcal{P}$. Then in s each process is in $\{E_R, R, W, S, D\}$ and there exists at least process in $\{W, S, D\}$. Let $\mathcal{A}$ be an adversary of *Fairadvs*, and let α be an extended execution of $\Omega_{prexec(M,\{s\},\mathcal{A})}$.

We first argue that eventually some process reaches a state of $\{S, D, F\}$ in α. This is trivially true if in state s there is some process in $\{S, D\}$. If this is not the case, then all processes are either in E_R or R or W. Eventually, some process in R or W performs a transition. If the first process not in E_R performing a transition started in E_R or R, then it reaches F and we are done; if the first process performing a transition is in W, then it reaches S since in s no resource is held. Once a process i is in $\{S, D, F\}$, then eventually process i reaches either state F or P, and we are done. ■

6.4 General Coin Lemmas

The coin lemmas of Section 6.2 are sufficiently general to prove the correctness of the Randomized Dining Philosophers algorithm of Lehmann and Rabin. However, there are several other coin events that are relevant for the analysis of distributed algorithms. For example, the toy resource allocation protocol that we used in Chapter 5 cannot be verified yet. In this section we present two general coin lemmas: the first one deals with multiple outcomes in a random draw; the second one gives a generalization of all the coin lemmas presented in the thesis. Unfortunately, generality and simplicity are usually incompatible: the two coin lemmas of this section are conceptually more complicated than those of Section 6.2.

6.4.1 Conjunction of Separate Coin Events with Multiple Outcomes

The coin lemma of Section 6.2.4 deals with the result of the intersection of several coin events. Thus, for example, if each coin event expresses the process of flipping a coin, then the coin lemma of Section 6.2.4 can be used to study the probability that all the coins yield head.

However, we may be interested in the probability that at least half of the coins yield head, or in the probability that exactly 5 coins yield head. The coin lemmas of Section 6.2 are not adequate. Suppose now that we use each coin event to express the process of rolling a dice. The coin events of Section 6.2 are not adequate again since they can deal only with binary outcomes: we can observe only whether a specific set U is reached or not. How can we express the event that for each number i between 1 and 6 there is at least one dice that rolls i?

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

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

Then define $GFIRST(\mathcal{S}, E)$ to be the function that associates with each probabilistic execution fragment H of M the set of extended executions α of Ω_H such that $E \cap (U_1(\alpha \triangleright q_0^H) \times \cdots \times U_k(\alpha \triangleright q_0^H)) \neq \emptyset$.

We illustrate the definition above by encoding the dice rolling example. In each tuple $(a_i, U_{i,1}, \ldots, U_{i,k})$ a_i identifies the action of rolling the i^{th} dice, $k = 6$, and for each j, $U_{i,j}$ is the set of states where the i^{th} dice rolls j. The set E identifies the set of outcomes that are considered to be good. In the case of the dices E is the set of tuples $((1, j_1), \ldots, (n, j_n))$ where for each number l between 1 and 6 there is at least one i such that $j_i = l$. The function $U_i(\alpha)$ checks whether the i^{th} dice is rolled and identifies the outcome. If the dice is not rolled, then, we allow any outcome as a possible one; if the dice is rolled and hits $U_{i,j}$, then the outcome is (i, j); if the the dice is rolled and the outcome is not in any one of the sets $U_{i,j}$'s, then there is no outcome (this case does not arise in our example). Then, an extended execution α of Ω_H is in the event $GFIRST(\mathcal{S}, E)(H)$ if at least one of the outcomes associated with $\alpha \triangleright q_0^H$ is an element of E, i.e., if by choosing the outcome of the dices that are not rolled in $\alpha \triangleright q_0^H$ all the six numbers appear as the outcome of some dice.

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

Proposition 6.4.1 *Let M be a probabilistic automaton. Let $\mathcal{S}$ be a set of n tuples $\{x_1, \ldots, x_n\}$ where for each i, $1 \leq i \leq n$, x_i is a tuple $(a_i, U_{i,1}, \ldots, U_{i,k})$ consisting of an action of M and k pairwise disjoint sets of states of M. Let the actions a_i be all distinct. Let E be a set of tuples $((1, j_1), \ldots, (n, j_n))$ where for each i, $1 \leq i \leq n$, the value of j_i is between 1 and k. For each i, j, $1 \leq i \leq n$, $1 \leq j \leq k$, let $p_{i,j}$ be a real number between 0 and 1 such that for each transition $(s, \mathcal{P})$ of M where $P[a_i] > 0$, $P[U_{i,j} | a_i] \geq p_{i,j}$, and let $\mathcal{C}$ be the collection of the $p_{i,j}$s. Let $P_{\mathcal{C}}[E]$ be the probability of the event E assuming that each experiment i is run independently, and that for each i a pair (i, j) is chosen with probability $p_{i,j}$. Then, for each probabilistic execution fragment H of M, $P_H[GFIRST(\mathcal{S}, E)(H)] \geq P_{\mathcal{C}}[E]$.*

Proof. For each state q of H, each $i \in \{1, \ldots, n\}$, and each $j \in \{1, \ldots, k\}$, denote by $\Omega(q, U_{i,j})$ the set $\{(a_i, q') \in \Omega_q^H \mid lstate(q') \in U_{i,j}\}$ of pairs where a_i occurs and leads to a state of $U_{i,j}$, and denote by $\Omega(q, \overline{U_i})$ the set $\{(a_i, q') \in \Omega_q^H \mid \mathit{lstate}(q') \notin \cup_j U_{i,j}\}$ of pairs where a_i occurs and none of the $U_{i,j}$s is reached. For each $i \in \{1, \ldots, n\}$, let Θ_i be the set of states q of H such that no action a_j, $1 \leq j \leq n$, occurs in $q \triangleright q_0^H$, and $P_q^H[a_i] > 0$.

We prove the lemma by induction on n. If $n = 1$ then the result follows from Lemma 6.2.1 (the event can be transformed into a new event with two outcomes); otherwise,

$$\begin{aligned} P_H[\overline{\mathit{GFIRST}(\mathcal{S}, E)(H)}] = \sum_{i\in\{1,\ldots,n\}} \sum_{q\in\Theta_i} P_H[C_q] \left(\left(\sum_{(a_i,q')\in\Omega(q,\overline{U_i})} P_q^H[(a_i, q')] \right) \right. \\ + \left. \left(\sum_{j\in\{1,\ldots,k\}} \sum_{(a_i,q')\in\Omega(q,U_{i,j})} P_q^H[(a_i, q')] P_{H\triangleright q'}[\overline{\mathit{GFIRST}(\mathcal{S}_i, E_{(i,j)})(H\triangleright q')}] \right) \right). \end{aligned} \tag{6.23}$$

where $\mathcal{S}_i$ is obtained from $\mathcal{S}$ by removing the tuple $(a_i, U_{i,1}, \ldots, U_{i,k})$, and $E_{(i,j)}$ is the set of tuples $((1, j_1), \ldots, (i-1, j_{i-1}), (i+1, j_{i+1}), \ldots, (n, j_n))$ such that $((1, j_1), \ldots, (i-1, j_{i-1}), (i, j), (i+1, j_{i+1}), \ldots, (n, j_n)) \in E$. Let $\mathcal{C}_i$ be obtained from $\mathcal{C}$ by removing all the probabilities of the form $p_{i,j}$, $1 \leq j \leq k$. Then, by induction,

$$P_{H\triangleright q'}[\overline{\mathit{GFIRST}(\mathcal{S}_i, E_{(i,j)})(H\triangleright q')}] \leq (1 - P_{\mathcal{C}_i}[E_{(i,j)}]). \tag{6.24}$$

From the properties of conditional probabilities and the definition of $\mathcal{C}$,

$$P_{\mathcal{C}_i}[E_{(i,j)}] = P_{\mathcal{C}}[E|(i, j)]. \tag{6.25}$$

Thus, by using (6.24) and (6.25) in (6.23), and by expressing $P_q^H[(a_i, q')]$ as $P_q^H[a_i] P_q^H[(a_i, q')|a_i]$, we obtain

$$\begin{aligned} P_H[\overline{\mathit{GFIRST}(\mathcal{S}, E)(H)}] \leq \sum_{i\in\{1,\ldots,n\}} \sum_{q\in\Theta_i} P_H[C_q] P_q^H[a_i] \left(\left(\sum_{(a_i,q')\in\Omega(q,\overline{U_i})} P_q^H[(a_i, q')|a_i] \right) \right. \\ + \left. \left(\sum_{j\in\{1,\ldots,k\}} \sum_{(a_i,q')\in\Omega(q,U_{i,j})} P_q^H[(a_i, q')|a_i](1 - P_{\mathcal{C}}[E|(i, j)]) \right) \right). \end{aligned} \tag{6.26}$$

For each i, j and q, let $p_{i,j,q}$ be $P_q^H[\Omega(q, U_{i,j})|a_i]$. Then, from (6.26),

$$\begin{aligned} P_H[\overline{\mathit{GFIRST}(\mathcal{S}, E)(H)}] \leq \sum_{i\in\{1,\ldots,n\}} \sum_{q\in\Theta_i} P_H[C_q] P_q^H[a_i] \\ \left((1 - p_{i,1,q} - \cdots - p_{i,k,q}) + \left(\sum_{j\in\{1,\ldots,k\}} p_{i,j,q}(1 - P_{\mathcal{C}}[E|(i, j)]) \right) \right), \end{aligned} \tag{6.27}$$

which becomes

$$\begin{aligned} & P_H[\overline{\mathit{GFIRST}(\mathcal{S}, E)(H)}] \\ & \leq \sum_{i\in\{1,\ldots,n\}} \sum_{q\in\Theta_i} P_H[C_q] P_q^H[a_i] \left(1 - \sum_{j\in\{1,\ldots,k\}} P_{\mathcal{C}}[E|(i, j)] p_{i,j,q} \right) \end{aligned} \tag{6.28}$$

after some simple algebraic simplifications. Using the same argument as in the proof of Lemma 6.2.1, for each i, j and each q, $p_{i,j,q} \geq p_{i,j}$. Thus,

$$P_H[\overline{GFIRST(\mathcal{S}, E)(H)}] \\ \leq \sum_{i\in\{1,\ldots,n\}} \sum_{q\in\Theta_i} P_H[C_q]P_q^H[a_i]\left(1 - \sum_{j\in\{1,\ldots,k\}} P_{\mathcal{C}}[E|(i,j)]p_{i,j}\right). \tag{6.29}$$

Finally, observe that $\sum_{i\in\{1,\ldots,n\}}\sum_{q\in\Theta_i} P_H[C_q]P_q^H[a_i]$ is the probability that some action a_i occurs, and observe that $\sum_{j\in\{1,\ldots,k\}} P_{\mathcal{C}}[E|(i,j)]p_{i,j} = P_{\mathcal{C}}[E]$. Thus,

$$P_H[\overline{GFIRST(\mathcal{S}, E)(H)}] \leq 1 - P_{\mathcal{C}}[E] \tag{6.30}$$

■

6.4.2 A Generalized Coin Lemma

All the coin lemmas that we have studied in this chapter share a common characteristic. Given a probabilistic execution fragment H, we identify n separate classes of random draws to observe. Each class can be observed at most once in every execution α of Ω_H, and if any class cannot be observed, then we allow for any arbitrary outcome. In this section we formalize this idea.

Let H be a probabilistic execution fragment of a probabilistic automaton M. A *coin-event specification* for H is a collection C of tuples $(q, X, X_1, \ldots, X_k)$ consisting of a state of H, a subset X of Ω_q^H, and m pairwise disjoint subsets of X, such that the following properties are satisfied:

1. for each state q of H there is at most one tuple of C whose state is q;

2. for each state q of H such that there exists a tuple of C with state q, there is no prefix q' of q such that there exists a tuple $(q', X, X_1, \ldots, X_k)$ in C and a pair (a, q'') in X where q'' is a prefix of q.

The set C is the object that identifies one of the classes of random draws to be observed. For each transition tr_q^H and each tuple $(q, X, X_1, \ldots, X_k)$ of C, the set X identifies the part of tr_q^H that is relevant for C, and the sets $X_1, \ldots, X_k$ identify some of the possible outcomes. The first requirement for C guarantees that there is at most one way to observe what happens from a state q of H, and the second requirement states that along every execution of Ω_H there is at most one place where C is observed.

As an example, consider the observation of whether the first occurrence of an action a, which represents a coin flip, leads to head. Then C is the set of tuples (q, X, X_1) where action a does not occur in $q \triangleright q_0^H$ and $P_q^H[a] > 0$, X is the set of pairs of Ω_q^H where action a occurs, and X_1 is the set of pairs of X where the coin flips head.

Let α be an extended execution of Ω_H, and let q be a state of H such that $q \leq \alpha$. We say that C *occurs* in α at q iff there exists a tuple $(q, X, X_1, \ldots, X_k)$ in C and a pair (a, q') in X such that $q' \leq \alpha$. Moreover, if $(a, q') \in X_j$, we say that C occurs in α at q and leads to X_j.

Two coin event specifications C_1 and C_2 are said to be *separate* iff from every state q of H, if $(q, X_1, X_{1,1}, \ldots, X_{1,k})$ is a tuple of C_1 and $(q, X_2, X_{2,1}, \ldots, X_{2,k})$ is a tuple of C_2, then $X_1 \cap X_2 = \emptyset$. In other words, there is no interference between the observations of C_1 and the observations of C_2. Let $\mathcal{S} = \{C_1, \dots, C_n\}$ be a set of pairwise *separate* coin-event specifications. For notational convenience, for each $i \in \{1, \dots, n\}$ and each state q of H such that there exists a tuple in C_i with state q, denote such tuple by $(q, X_{q,i}, X_{q,i,1}, \dots, X_{q,i,k})$

Let E be a set of tuples $((1, j_1), \dots, (n, j_n))$ where for each i, $1 \leq i \leq n$, the value of j_i is between 1 and k. For each extended execution α of Ω_H and each i, $1 \leq i \leq n$, let

$$U_i(\alpha) = \begin{cases} \{(i,1), \dots, (i,k)\} & \text{if } C_i \text{ does not occur in } \alpha \\ \{(i,j)\} & \text{if } C_i \text{ occurs in } \alpha \text{ leading to } X_{q,i,j} \\ \emptyset & \text{otherwise.} \end{cases}$$

Then, define $GCOIN(\mathcal{S}, E)(H)$ to be the set of extended executions of Ω_H such that $E \cap (U_1(\alpha \triangleright q_0^H) \times \cdots \times U_k(\alpha \triangleright q_0^H)) \neq \emptyset$.

Lemma 6.4.2 *Let H be a probabilistic execution fragment of a probabilistic automaton M. Let $\mathcal{S} = \{C_1, \dots, C_n\}$ be a set of separate coin-event specifications for H. For each i, j, $1 \leq i \leq n$, $1 \leq j \leq k$, let $p_{i,j}$ be a real number between 0 and 1 such that for each $i \in \{1, \dots, n\}$ and each tuple $(q, X_{q,i}, X_{q,i,1}, \dots, X_{q,i,m})$ of C_i, $P_q^H[X_{q,i,j} | X_{q,i}] \geq p_{i,j}$. Let $\mathcal{C}$ be the collection of the $p_{i,j}$'s. Let $P_{\mathcal{C}}[E]$ be the probability of the event E assuming that each experiment i is run independently, and for each i a pair (i,j) is chosen with probability $p_{i,j}$. Then, $P_H[GCOIN(\mathcal{S}, E)(H)] \geq P_{\mathcal{C}}[E]$.*

Proof. For each state q of H and each i, $1 \leq i \leq n$, if there exists a tuple in C_i with state q, then denote $X_{q,i} \setminus \cup_{j \in \{1, \dots, k\}} X_{q,i,j}$ by $\overline{X_{q,i}}$. For each i, $1 \leq i \leq n$, let Θ_i be the set of states q of H such that there exists a tuple with state q in C_i and no coin-event C_j, $1 \leq j \leq n$, occurs in $q \triangleright q_0^H$.

We prove the lemma by induction on n, using $n = 0$ for the base case. For $n = 0$ we assume that $P[E] = 1$ and that $GCOIN(\mathcal{S}, E)(H) = \Omega_H$. In this case the result is trivial. Otherwise,

$$\begin{aligned} P_H[\overline{GCOIN(\mathcal{S}, E)(H)}] = {} & \sum_{i \in \{1, \dots, n\}} \sum_{q \in \Theta_i} P_H[C_q] \left(\left(\sum_{(a,q') \in \overline{X_{q,i}}} P_q^H[(a, q')] \right) \right. \\ & + \left. \left(\sum_{j \in \{1, \dots, k\}} \sum_{(a,q') \in X_{q,i,j}} P_q^H[(a,q')] P_{H \triangleright q'}[\overline{GCOIN(\mathcal{S} \triangleright q', E_{(i,j)})(H \triangleright q')}] \right) \right). \end{aligned} \tag{6.31}$$

where $\mathcal{S} \triangleright q'$ is obtained from $\mathcal{S}$ by removing C_i and, for each $j \neq i$, by transforming the set C_j into $\{(q \triangleright q', X \triangleright q', X_1 \triangleright q', \dots, X_k \triangleright q') \mid (q, X, X_1, \dots, X_k) \in C_j, q' \leq q\}$. Then, by induction,

$$P_{H \triangleright q'}[\overline{GCOIN(\mathcal{S} \triangleright q', E_{(i,j)})(H \triangleright q')}] \leq (1 - P_{\mathcal{C}_i}[E_{(i,j)}]). \tag{6.32}$$

From the properties of conditional probabilities and the definition of $\mathcal{C}$,

$$P_{\mathcal{C}_i}[E_{(i,j)}] = P_{\mathcal{C}}[E | (i,j)]. \tag{6.33}$$

Thus, by using (6.32) and (6.33) in (6.31), and expressing $P_q^H[(a, q')]$ as $P_q^H[X_{q,i}] P_q^H[(a,q') | X_{q,i}]$, we obtain

$$\begin{aligned} P_H[\overline{GCOIN(\mathcal{S}, E)(H)}] \leq {} & \sum_{i \in \{1, \dots, n\}} \sum_{q \in \Theta_i} P_H[C_q] P_q^H[X_{q,i}] \left(\left(\sum_{(a,q') \in \overline{X_{q,i}}} P_q^H[(a, q') | X_{q,i}] \right) \right. \\ & + \left. \left(\sum_{j \in \{1, \dots, k\}} \sum_{(a,q') \in X_{q,i,j}} P_q^H[(a,q') | X_{q,i}] (1 - P_{\mathcal{C}}[E | (i,j)]) \right) \right). \end{aligned} \tag{6.34}$$

For each i, j and q, let $p_{i,j,q}$ be $P_q^H[X_{q,i,j}|X_{q,i}]$. Then, from (6.34),

$$P_H[\overline{GCOIN(\mathcal{S}, E)(H)}] \leq \sum_{i \in \{1,\dots,n\}} \sum_{q \in \Theta_i} P_H[C_q] P_q^H[X_{q,i}] \left((1 - p_{i,1,q} - \dots - p_{i,k,q}) + \left(\sum_{j \in \{1,\dots,k\}} p_{i,j,q}(1 - P_{\mathcal{C}}[E|(i,j)]) \right) \right), \tag{6.35}$$

which becomes

$$P_H[\overline{GCOIN(\mathcal{S}, E)(H)}] \leq \sum_{i \in \{1,\dots,n\}} \sum_{q \in \Theta_i} P_H[C_q] P_q^H[X_{i,j}] \left(1 - \sum_{j \in \{1,\dots,k\}} P_{\mathcal{C}}[E|(i,j)] p_{i,j,q} \right) \tag{6.36}$$

after some simple algebraic simplifications. From hypothesis, for each i, j and each q, $p_{i,j,q} \geq p_{i,j}$. Thus,

$$P_H[\overline{GCOIN(\mathcal{S}, E)(H)}] \leq \sum_{i \in \{1,\dots,n\}} \sum_{q \in \Theta_i} P_H[C_q] P_q^H[X_{q,i}] \left(1 - \sum_{j \in \{1,\dots,k\}} P_{\mathcal{C}}[E|(i,j)] p_{i,j} \right). \tag{6.37}$$

Finally, observe that $\sum_{i \in \{1,\dots,n\}} \sum_{q \in \Theta_i} P_H[C_q] P_q^H[X_{q,i}]$ is the probability that some C_i occurs, and observe that $\sum_{j \in \{1,\dots,k\}} P_{\mathcal{C}}[E|(i,j)] p_{i,j} = P_{\mathcal{C}}[E]$. Thus,

$$P_H[\overline{GCOIN(\mathcal{S}, E)(H)}] \leq 1 - P_{\mathcal{C}}[E] \tag{6.38}$$

■

6.5 Example: Randomized Agreement with Stopping Faults

In this section we analyze the Randomized Agreement algorithm of Ben-Or [BO83]. Its proof of correctness is an application of Lemma 6.4.2. The proof that we present in this section is not as detailed as the proof of the Dining Philosophers algorithm, but contains all the information necessary to fill in all the details, which we leave to the reader.

6.5.1 The Problem

Consider n asynchronous processes that communicate through a network of reliable channels (i.e., channels that deliver all the messages in the same order as they are received, and that never fail to deliver a message), and suppose that each process i starts with an initial value $v_i \in \{0, 1\}$. Suppose that each process can broadcast a message to every other process in a single operation. Each process runs an algorithm that at some point may decide on one value of $\{0, 1\}$. Each process decides at most once. The algorithm should be designed so that the following properties are satisfied.

1. **Agreement:** all the processes that decide choose the same value.

2. **Validity:** if all the processes have the same initial value v, then v is the only possible decision value.

3. **f-failure termination:** if at most f processes fail, then all the non-failing processes decide a value.

We assume that a process fails by stopping, i.e., by failing to send messages to other processes from some point on. Since the processes are asynchronous, no processes can distinguish a slow process from a failing process.

Unfortunately, it is known from [FLP85] that there is no deterministic algorithm for asynchronous processes that solves the agreement problem and guarantees 1-failure termination. Here we present the randomized algorithm of Ben-Or [BO83], which solves the agreement problem with certainty, and guarantees f-failure termination with probability 1 whenever $n > 3f$.

6.5.2 The Algorithm

Each process i has local variables x, initially v_i, and y, initially *null*, and executes a series of *stages* numbered $1, 2, \ldots$, each stage consisting of two *rounds*. Each process runs forever, even after it decides. At stage $st \geq 1$, process i does the following.

1. Broadcast (first, st, v), where v is the current value of x, and then wait to obtain $n - f$ messages of the form $(\mathit{first}, st, *)$, where $*$ stands for any value. If all the messages have the same value v, then set $y := v$, otherwise set $y := \mathit{null}$.

2. Broadcast (second, st, v), where v is the current value of y, and then wait to obtain $n - f$ messages of the form $(\mathit{second}, st, *)$. There are three cases:

   (a) if all the messages have the same value $v \neq \mathit{null}$, then set $x := v$ and perform a $\mathit{decide}(v)_i$ operation if no decision was made already;

   (b) if at least $n - 2f$ messages, but not all the messages, have the same value $v \neq \mathit{null}$, then set $x := v$ without deciding (the assumption $n > 3f$ guarantees that there cannot be two different such values v);

   (c) otherwise, set x to 0 with probability 1/2 and to 1 with probability 1/2.

The intuition behind the use of randomness is that at each stage, if a decision is not made yet, with probability at least $1/2^n$ all the processes that choose a value at random choose the same "good" value. Thus, with probability 1 there is eventually a stage where the processes that choose a value at random choose the same good value, and this leads to a decision.

We now give an idea of the structure of the probabilistic automaton M that describes Ben-Or's algorithm. Each process i has the two variables x and y mentioned in the description of the algorithm, plus a queue m_j for each process j that records the unprocessed messages received from process j, initially *null*, a stage counter st, initially 1, a program counter pc, and a boolean variable *decided* that is set to *true* iff process i has decided already. There is a channel $C_{i,j}$ between every pair of processes. Each channel $C_{i,j}$ is essentially a buffer like the buffer described in Chapter 3 (cf. Figure 3-1), whose inputs are actions of the form $(\mathit{first}, st, v)_i$ and $(\mathit{second}, st, v)_i$, and whose outputs are actions of the form $(\mathit{first}, st, v)_{i,j}$ and $(\mathit{second}, st, v)_{i,j}$. To broadcast a message (first, st, v), process i performs the action $(\mathit{first}, st, v)_i$.

A message (first, st, v) is received by process i from process j through the action $(\mathit{first}, st, v)_{j,i}$. The definition of the transition relation of M is straightforward.

6.5.3 The High Level Proof

Agreement and validity are easy to prove and do not involve any probabilistic argument.

Lemma 6.5.1 *Ben-Or's algorithm satisfies the agreement and validity conditions.*

Proof. We start with validity. Suppose that all the processes start with the same value v. Then it is easy to see that every process that completes stage 1 decides on v in that stage. This is because the only value sent or received by any process in the first round is v, and thus the only value sent or received by any process in the second round is v, leading to the decision of v.

For agreement, suppose that some process decides, and let process i be the first process that decides. Let v and st be the value decided by process i and the stage at which process i decides, respectively. Then it must be the case that process i receives $n-f$ (second, st, v) messages. This implies that any other process j that completes stage st receives at least $n-2f$ (second, st, v) messages, since it hears from all but at most f of the processes that process i hears from. This means that process j cannot decide on a value different from v at stage st; moreover, process j sets $x := v$ at stage st. Since this is true for all the processes that complete stage st, then an argument similar to the argument for validity shows that any process that completes stage $st+1$ and does not decide in stage st decides v at stage $st+1$. ■

The argument for f-failure termination involves probability. We assume that all the processes but at most f are scheduled infinitely many times. Thus, let *f-fair* be the set of adversaries for M such that for each probabilistic execution fragment H generated by an adversary of *f-fair* the set Ω_H contains only executions of M where at least $n-f$ processes are scheduled infinitely many times. It is easy to check that *f-fair* is finite-history-insensitive.

Let $\mathcal{B}$ be the set of reachable states of M; let $\mathcal{F}$ be the set of reachable states of M where no process has decided yet and there exists a value st and a number i such that process i received exactly $n-f$ messages $(\mathit{first}, st, *)$, and no other process has ever received more than $n-f-1$ messages $(\mathit{first}, st, *)$; finally, let $\mathcal{O}$ be the set of reachable states of M where at least one process has decided.

It is easy to show that

$$\mathcal{B} \xrightarrow[1]{}_{\mathit{f\text{-}fair}} \mathcal{F} \cup \mathcal{O}. \tag{6.39}$$

Specifically, let α be an f-fair execution fragment of M starting from a reachable state s of M, and let st be the maximum value of the stages reached by each process in s. Then, stage $st+1$ is reached eventually in α, and thus there is a state s' in α where some process is the first one to receive $n-f$ messages $(\mathit{first}, st+1, *)$. The state s' is a state of $\mathcal{F} \cup \mathcal{O}$.

In Section 6.5.4 we show that

$$\mathcal{F} \xrightarrow[1/2^n]{} \mathcal{O}. \tag{6.40}$$

Thus, combining (6.39) and (6.40) with Theorem 5.5.2, and by using Proposition 5.5.6, we obtain

$$\mathcal{B} \xrightarrow[1]{} \mathcal{O}. \tag{6.41}$$

Finally, we need to show that in every f-fair execution where at least one process decides all the non-failing processes decide eventually. This is shown already in the second part of the proof of Lemma 6.5.1.

6.5.4 The Low Level Proof

In this section we prove the progress statement of (6.40) using the generalized coin lemma. Consider a state s of $\mathcal{F}$, and let i be the process that has received $n-f$ messages (first, st, v). Let $\mathcal{A}$ be an adversary of *f-fair*, and let H be $\mathit{prexec}(M, \mathcal{A}, s)$.

For each j, $1 \leq j \leq n$, let C_j be the set of triplets (q, X, X_1) where q is a state of H such that process j is at stage st in $\mathit{lstate}(q)$ and there is a non-zero probability that process j chooses randomly between 0 and 1 from q, X is the set of pairs of Ω_q^H where process j performs a transition, and X_1 is defined as follows. Let s' be $\mathit{lstate}(q)$, and let v be a *good* value if at least $f+1$ of the messages $(\mathit{first}, st, *)$ processed by process i have value v. We emphasize the word "processed" since, although each process can receive more that $n-f$ messages $(\mathit{first}, st, *)$, only $n-f$ of those messages are used (processed).

1. If 0 is a good value, then let X_1 be the set of pairs of X where process i chooses 0;

2. if 1 is a good value and 0 is not a good value, then let X_1 be the set of pairs of X where process i chooses 1.

Observe that in s' there is at least one good value, and at most two values; thus, C_j is well defined. It is easy to check that $C_1, \ldots, C_n$ are separate coin event specifications; moreover, for each j, $1 \leq j \leq n$, and each triplet (q, X, X_1) of C_j, $P_q^H[X_1|X] = 1/2$. Let $E = \{((1,1),(2,1),\ldots,(n,1)\}$. From Lemma 6.4.2, $P_H[\mathit{GCOIN}((C_1,\ldots,C_n),E)(H)] \geq 1/2^n$.

We are left with the proof that in each extended execution of $\mathit{GCOIN}((C_1,\ldots,C_n),E)(H)$ all the non-faulty processes choose a value. More precisely, we show that the non-faulty processes complete stage st setting x to the same value v. Then, the second part of the proof of Lemma 6.5.1 can be used to show that all the non-faulty processes decide on v at the end of stage $st+1$; in particular at least one process decides. We distinguish two cases.

1. In s' there is exactly one good value v.

   In this case every other process receives at least one copy of v during the first round of stage st, and thus y is set either to v or to *null*. Therefore, v is the only value that a process chooses by a non-random assignment at the end of stage st. On the other hand, if a process j chooses a value at random at the end of stage st, the definition of C_j guarantees that the value chosen is v. Thus, every process that completes stage st sets $x := v$.

2. In s' there are two good values.

   In this case every process receives at least one copy of 0 and one copy of 1, and thus y is set to *null*. Therefore, each process chooses a value at random at the end of stage st. The definition of $C_1, \ldots, C_n$ guarantees that every process that completes stage st sets $x := 0$.

6.6 Example: The Toy Resource Allocation Protocol

Lemma 6.4.2 can be used also to prove formally that the toy resource allocation protocol of Section 5.1 guarantees that, under any deterministic fair oblivious adversary (cf. Example 5.6.2 for the definition of a fair oblivious adversary), process M_1 eventually gets a resource. This result can be extended to general oblivious adversaries by using the results about deterministic and randomized adversaries proved in Chapter 5 (cf. Proposition 5.7.11).

Recall from Example 6.1.1 that we want to identify a coin event that expresses the following property: the first coin flip of M_1 after the first coin flip of M_2 is different from the last coin flip of M_2 before the first time M_1 checks its resource after flipping. In the rest of the section we specify two coin event specifications C_1 and C_2. The specification C_1 identifies the first coin flip of M_1 after the first coin flip of M_2, while the specification C_2 identifies the last coin flip of M_2 before the first time M_1 checks its resource after flipping.

Let H be a probabilistic execution fragment, generated by a deterministic fair oblivious adversary, such that the first state of q_0^H is reachable in M. Let C_1 be the set of tuples (q, X, X_1, X_2) where

1. q is a state of H such that M_2 flips at least once in $q \triangleright q_0^H$, M_1 does not flip in $q \triangleright q_0^H$ after the first time M_2 flips, and M_1 flips from q,
2. X is the set Ω_q^H,
3. X_1 is the set of pairs of X where M_1 flips head,
4. X_2 is the set of pairs of X where M_1 flips tail.

Observe that C_1 is a coin-event specification. Moreover, observe that for each tuple of C_1, $P_q^H[X_1|X] = 1/2$ and $P_q^H[X_2|X] = 1/2$. Let C_2 be the set of tuples (q, X, X_1, X_2) where

1. q is a state of H such that either
   (a) M_1 does not flip in $q \triangleright q_0^H$ after M_2 flips, M_2 flips from q, and there exists a state $q' \geq q$ such that M_2 flips exactly once in $q' \triangleright q$ and M_1 flips and checks its resource after flipping in $q' \triangleright q$, or
   (b) M_1 flips and does not check its resource after the first flip of M_2 in $q \triangleright q_0^H$, M_2 flips from q, and there exists a state $q' \geq q$ such that M_2 flips exactly once in $q' \triangleright q$, M_1 does not check its resource in $q' \triangleright q$, and M_1 checks its resource from q',
2. X is the set Ω_q^H,
3. X_1 is the set of pairs of X where M_2 flips head,
4. X_1 is the set of pairs of X where M_2 flips tail.

Informally, C_2 identifies the coin flip of M_2 that precedes the point where M_1 checks the resource determined by C_1. Figure 6-4 illustrates graphically the two cases of the definition of C_2. Observe that for each tuple of C_2, $P_q^H[X_1|X] = 1/2$ and $P_q^H[X_2|X] = 1/2$. Since H is generated by an oblivious deterministic adversary, then it is easy to verify that C_2 is a coin-event specification. The important point is to verify that Condition 2 of the definition of a coin event is satisfied; this is the point where the fact that an adversary is oblivious and deterministic is used.

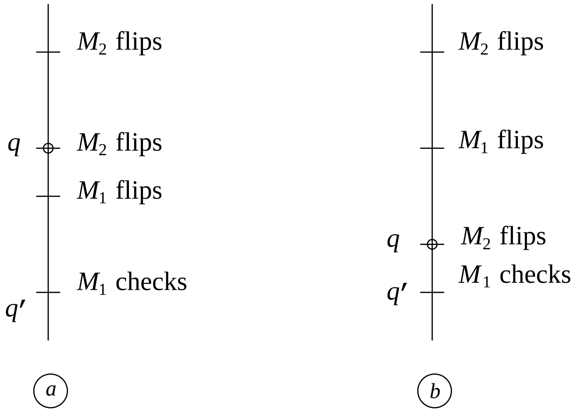


Figure 6-4: The definition of C_2 for the toy resource allocation protocol.

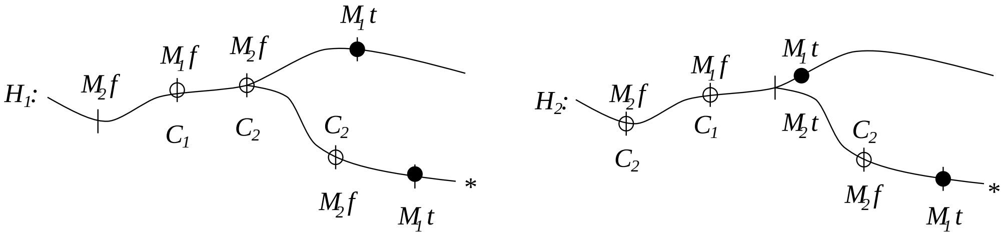


Figure 6-5: How C_2 could not be a coin event specification.

Example 6.6.1 (How C_2 could not be a coin event specification.) To give a rough idea of why Condition 2 does not fail, Figure 6-5 shows how Condition 2 could fail. Consider the execution of H_1 that is marked with $*$, and denote it by α; denote by α' the other execution of H_1 that appears in the figure. The unfilled circles mark the points where a coin event specification is observed. By following α from left to right we observe C_1 and then we observe C_2. The reason why we observe C_2 the first time is that along α' M_1 tests its resource. However, continuing to follow α, we observe C_2 again because along α M_2 tests its resource later. Using oblivious adversaries we are guaranteed that such a situation does not arise because if along α' M_1 tests its resource before M_2 flips again, then the same property holds along α.

The probabilistic execution H_2 of Figure 6-5 illustrates how Condition 2 can fail by using randomized schedulers. After M_1 flips, the adversary chooses randomly whether to let M_1 test its resource (higher filled circle) or to let M_2 continue. ■

Let E be the set $\{((1,1)(2,2)),((1,2),(2,1))\}$, which expresses the fact that C_1 and C_2 yield two different outcomes. It is easy to check that in every execution of $GCOIN((C_1,C_2),E)(H)$ M_1 eventually gets one resource. Thus, from Lemma 6.4.2, the probability that M_1 gets its resource in H is at least 1/4. Since H is a generic probabilistic execution fragment, then, under any deterministic fair oblivious adversary M_1 gets a resource eventually with probability at least 1/4. Since the set of deterministic fair oblivious adversaries is finite-history-insensitive, Lemma 5.5.6 applies, and we conclude that under any deterministic fair oblivious adversary M_1 gets a resource eventually with probability 1.

6.7 The Partition Technique

Even though the coin lemmas can be used to prove the correctness of several nontrivial algorithms, two of which have been illustrated in this chapter, there are algorithms for which the coin lemmas do not seem to be suitable. One example of such an algorithm is the randomized algorithm for maximal independent sets of Awerbuch, Cowen and Smith [ACS94]; another example is the toy resource allocation protocol again.

Example 6.7.1 (The coin lemmas do not work always) In Section 6.6 we have shown that the toy resource allocation protocol guarantees progress against fair oblivious adversaries; however, in Example 5.6.2 we have stated that the toy resource allocation protocol guarantees progress also against adversaries that do not know only the outcome of those coins that have not been used yet. Such a result cannot be proved using the coin lemmas of this chapter because situations like those outlined in Example 6.6.1 arise. For example, after the first time M_2 flips, we could schedule M_2 again and then schedule M_1 to test its resource only if M_2 gets the resource R_1.

Another way to obtain a situation where the coin lemmas of this chapter do not apply is to modify the second instruction of the resource allocation protocol as follows

2. if the chosen resource is free, then get it, *otherwise go back to 1.* ■

Example 6.7.1 shows us that some other techniques need to be developed; it is very likely that several new techniques will be discovered by analyzing other algorithms. In this section we hint at a proof technique that departs considerably from the coin lemmas and that is sufficiently powerful to deal with the toy resource allocation protocol. We illustrate the technique with an example.

Example 6.7.2 (The partition technique) Let $\mathcal{A}$ be a generic fair adversary for the toy resource allocation protocol that does not know the outcome of those coin flips that have not been used yet, and let H be a probabilistic execution generated by $\mathcal{A}$. Assume for simplicity that $\mathcal{A}$ is deterministic; the result for a generic adversary follows from Proposition 5.7.11. Consider an element of Ω_H, and consider the first point q where M_1 flips a coin (cf. Figure 6-6). The coin flipping transition leads to two states q_h and q_t that are not distinguishable by $\mathcal{A}$, which means that from q_h and q_t the adversary schedules the same process. If the process scheduled from q_h and q_t is M_2, then the states reached from q_h are in one-to-one correspondence with the states reached from q_t, since they differ only in the value of the coin flipped by M_1. Figure 6-6 illustrates the case where M_2 flips a coin. Furthermore, two corresponding states are reached with the same probability. The one-to-one correspondence between the states reached form q_h and q_t is maintained until M_1 tests its chosen resource.

Consider now a point where M_1 tests its resource. Figure 6-6 illustrates four of these points, denoted by $q_{t,1}$, $q_{h,1}$, $q_{t,2}$, and $q_{h,2}$. If M_1 fails to obtain the resource, it means that M_2 holds that resource at that point. However, M_2 holds the same resource in the corresponding state via the one-to-one correspondence M_2, while M_1 tests the other resource. Thus, M_1 succeeds in getting the chosen resource. (cf. states $q_{t,1}$ and $q_{h,1}$ of Figure 6-6.

The bottom line is that we have partitioned the states where M_1 checks its resource in two sets, and we have shown that for each pair of corresponding states there is at least one state where M_1 succeeds in getting a resource. In some cases, like for states $q_{t,2}$, and $q_{h,2}$ of

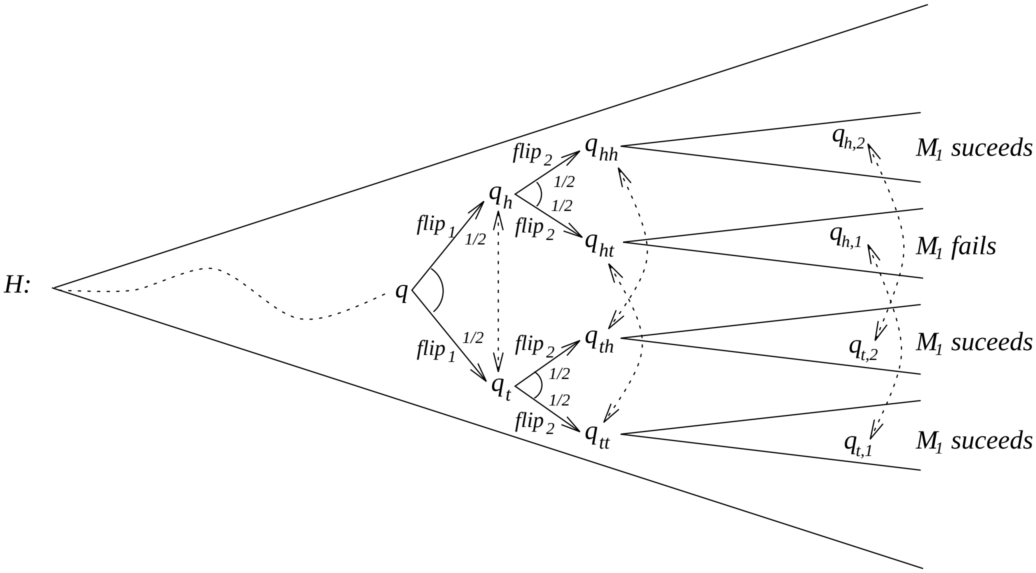


Figure 6-6: The partition technique.

Figure 6-6, M_1 succeeds in getting its resource from both the corresponding states (M_2 does not hold any resource). Thus, M_1 gets a resource with probability at least 1/2. ■

6.8 Discussion

To our knowledge, no techniques similar to our coin lemmas or to our partition technique were proposed before; however, similar arguments appear in several informal analysis of randomized algorithms. The idea of reducing the analysis of a randomized algorithm to the analysis of an ordinary pure nondeterministic system was at the base of the qualitative analysis techniques described in Sections 2.5.1 and 2.5.2. Here we have been able to apply the same idea for a quantitative analysis of an algorithm.

In this chapter we have focused mainly on how to apply a coin lemma for the verification of a randomized algorithm; once a good coin event is identified, the analysis is reduced to verify properties of a system that does not contain randomization. We have carried out this last part using detailed operational arguments, which can be error prone themselves. However, since the problem is reduced to the analysis of a non-randomized system, several existing techniques can be used to eliminate our operational arguments. In [PS95] Segala and Pogosyants show how such an analysis can be carried out formally and possibly mechanized.

Chapter 7

Hierarchical Verification: Trace Distributions

7.1 Introduction

So far we have defined a model to describe randomized concurrent and distributed systems, and we have shown how to study the properties of a system by means of a direct analysis of its structure. A specification is a set of properties that an implementation should satisfy, and an implementation is a probabilistic automaton that satisfies the desired properties.

Another approach to the analysis of a system considers an automaton as a specification itself. Then, an abstract notion of *observation* is defined on automata, and an automaton is said to be an implementation of another automaton iff there is a specific relation, usually a preorder relation, between their abstract observations. Examples of observations are traces [Hoa85, LV91] (cf. Section 3.2.3), and failures [Hoa85, BHR84]; in these two cases implementation is expressed by set inclusion.

7.1.1 Observational Semantics

Formally, an automaton A is associated with a set $Obs(A)$ of observations, and a preorder relation $\mathcal{R}$ is defined over sets of observations (for example $\mathcal{R}$ can be set inclusion). Then, an automaton A_1 is said to implement another automaton A_2, denoted by $A_1 \sqsubseteq A_2$, iff $Obs(A_1)\ \mathcal{R}\ Obs(A_2)$. The function $Obs()$ is called an *observational semantics*, or alternatively a *behavioral semantics*; in the second case the observations are thought as the possible behaviors of an automaton.

The methodology based on preorder relations is an instance of the hierarchical verification method: a specification, which is usually very abstract, can be refined successively into less abstract specifications, each one implementing the more abstract specification, till the actual implementation is obtained. Figure 7-1 gives an example of a specification that is refined two times to build the actual implementation. Of course it is implicitly assumed that the relevant properties of a system are only those that are preserved by the chosen implementation relation. Thus, given a relation, it is important to understand what properties it preserves. Coarse relations may not preserve all the relevant properties, but they are usually easy to verify, i.e., it is usually easy to establish whether such a relation holds; finer relations that preserve exactly the

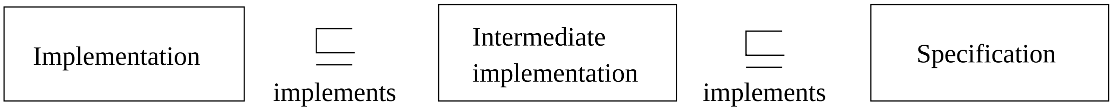


Figure 7-1: Refinement of a specification.

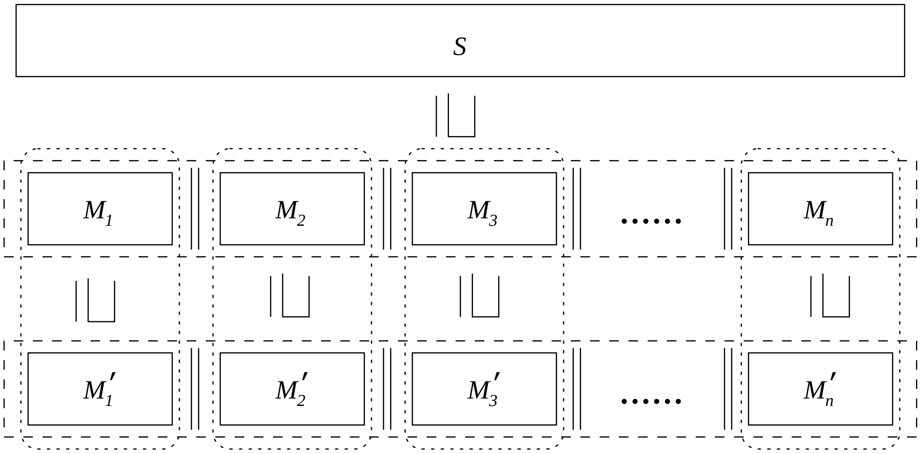


Figure 7-2: Modular design.

relevant properties are usually difficult to characterize and verify; other relations that preserve all the relevant properties and that are easy to verify are usually too fine, i.e., they distinguish too much. Some tradeoff is necessary.

7.1.2 Substitutivity and Compositionality

When the size of a problem becomes large, it is common to decompose the problem into simpler subproblems that are solved separately. Figure 7-2 gives an example. A large specification S is decomposed into several subcomponents $M_1, \ldots, M_n$ that interact together to implement S. For example, a complex computer system can be described by the interaction of a central processor unit, a memory unit, and an Input/Output unit. Then, each subcomponent specification M_i is given to a development team that builds an implementation M_i'. Finally, the implementations are put together to build an actual implementation of S. This kind of approach is called *modular design*; however, in order to guarantee the soundness of modular design, we need to guarantee that an implementation works properly in every context where its specification works properly, i.e., our implementation relation must be preserved by parallel composition (i.e., it must be a *precongruence*). This property is called *substitutivity* of a preorder relation, and constitutes one of the most important properties that an implementation relation should satisfy.

A property that is strictly related to the substitutivity of $\sqsubseteq$ is called *compositionality* of $Obs()$. That is, there is an operator $\|$ defined on pairs of sets of observations such that $Obs(A_1 \| A_2) = Obs(A_1) \| Obs(A_2)$. Compositionality and substitutivity are used interchangeably when talking informally about concurrent systems, and it is easy to get confused by the meanings of the two terms. To clarify every doubt, here is how the two concepts are related.

Theorem 7.1.1 *Let $Obs()$ be an observational semantics, $\mathcal{R}$ be an equivalence relation over sets of observations, and let, for each set x of observations, $[x]_{\mathcal{R}}$ be the equivalence class of x under $\mathcal{R}$. Let $A_1 \equiv A_2$ iff $Obs(A_1)\ \mathcal{R}\ Obs(A_2)$. Then the following two statements are equivalent.*

1. *$\equiv$ is substitutive, i.e., if $A_1 \equiv A_2$ then for each A_3, $A_1 \| A_3 \equiv A_2 \| A_3$;*

2. *$Obs()$ is compositional, i.e., there exists an operator $\|$ on equivalence classes of observations such that $[Obs(A_1 \| A_2)]_{\mathcal{R}} = [Obs(A_1)]_{\mathcal{R}} \| [Obs(A_1)]_{\mathcal{R}}$.* ■

If $\mathcal{R}$ is set equality, then we can remove the equivalence classes from the second statement since each set of observations is an equivalence class. The substitutivity of a preorder relation is stronger than the substitutivity of its kernel equivalence relation, since the direction of the inequality must be preserved under parallel composition. For this reason our primary concern in this chapter is the substitutivity of the implementation relation.

7.1.3 The Objective of this Chapter

In this chapter we study the simplest implementation relation based on observations, i.e., trace inclusion, and we extend the corresponding precongruence to the probabilistic framework. The trace preorder constitutes the basis for several other implementation relations and is known to preserve the *safety* properties of a system [AS85]. Roughly speaking, a safety property says that "something good holds forever" or that "something bad does not happen". The trace preorder is important for ordinary automata for its simplicity and for the availability of the *simulation method* [LT87, Jon91, LV91] (cf. Chapter 8), which provides several sufficient conditions for the trace preorder relation to hold. Other relations, based either on failures [Hoa85, BHR84] or on any other form of enriched traces, can be obtained by following the same methodology that we present here.

In the probabilistic framework a trace is replaced by a *trace distribution*, where the trace distribution of a probabilistic execution fragment H is the distribution over traces induced by $\mathcal{P}_H$, the probability space associated with H. The trace distribution preorder is defined as inclusion of trace distributions.

Unfortunately, the trace distribution preorder is not a precongruence (cf. Example 7.4.1), which in turn means that the observational semantics based on trace distributions is not compositional. A standard approach in this case is to define the *trace distribution precongruence* as the coarsest precongruence that is contained in the trace distribution preorder; then, in order to have a compositional observational semantics that captures the trace distribution precongruence, an alternative, more operational and constructive characterization of the trace distribution precongruence is derived. We give an alternative characterization of the trace distribution precongruence by exhibiting a context, called the *principal context*, that distinguishes two probabilistic automata whenever there exists a distinguishing context. This leads to the notion of a *principal trace distribution*, which is a trace distribution of a probabilistic automaton in parallel with the principal context; the trace distribution precongruence can be characterized alternatively as inclusion of principal trace distributions.

Several other characterizations of the trace distribution precongruence could be found, possibly leading to different observational semantics equivalent to the principal trace distribution semantics. Further experience with each one of the alternative semantics will determine which

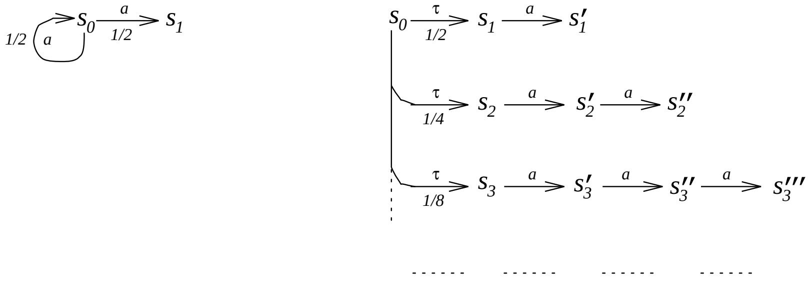


Figure 7-3: Trace distribution equivalent probabilistic automata.

one is more useful. One of the problems with the principal trace distribution characterization is that, although from Theorem 7.1.1 there exists an operator $\|$ defined on principal traces, the definition of $\|$ is not simple. For ordinary automata the traces of a parallel composition of two automata are exactly those sequences of actions that restricted to each component give a trace of the component. This property does not hold for principal trace distributions (cf. Example 7.4.1). It is desirable to find a semantics that characterizes the trace distribution precongruence and for which the corresponding parallel composition operator has a simple definition; however, it is not clear whether such a semantics exists.

7.2 Trace Distributions

Let H be a probabilistic execution fragment of a probabilistic automaton M, and let f be a function from Ω_H to $\Omega = ext(H)^* \cup ext(H)^\omega$ that assigns to each execution of Ω_H its trace. The *trace distribution* of H, denoted by $tdistr(H)$, is the probability space $completion((\Omega, \mathcal{F}, P))$ where $\mathcal{F}$ is the σ-field generated by the cones C_β, where β is a finite trace of H, and $P = f(P_H)$. Observe that, from Proposition 3.1.4, f is a measurable function from $(\Omega_H, \mathcal{F}_H)$ to $(\Omega, \mathcal{F})$, since the inverse image of a cone is a union of cones. Denote a generic trace distribution by $\mathcal{D}$. A trace distribution of a probabilistic automaton M is the trace distribution of one of the probabilistic executions of M. Denote by $tdistrs(M)$ the set of the trace distributions of a probabilistic automaton M.

It is easy to see that trace distributions extend the traces of ordinary automata: the trace distribution of a linear probabilistic execution fragment α is a distribution that assigns probability 1 to $trace(\alpha)$.

Given two probabilistic execution fragments H_1 and H_2, it is possible to check whether $tdistr(H_1) = tdistr(H_2)$ just by verifying that $P_{tdistr(H_1)}[C_\beta] = P_{tdistr(H_2)}[C_\beta]$ for each finite sequence of actions β. This is an easy consequence of the extension theorem (cf. Theorem 3.1.2).

Example 7.2.1 (Reason for the definition of Ω) The reader may wonder why we have not defined Ω to be $trace(\Omega_H)$. This is to avoid to distinguish two trace distribution just because they have different sample spaces. Figure 7-3 illustrates the idea. The two probabilistic automata of Figure 7-3 have the same trace distributions; however, the left probabilistic automaton has a probabilistic execution where the trace a^∞ occurs with probability 0, while the right probabilistic automaton does not. Thus, by defining the sample space of $tdistr(H)$ to be $trace(\Omega_H)$, the two probabilistic automata of Figure 7-3 would be distinct. In Chapter 8 we define several simulation relations for probabilistic automata, and we show that they are sound for the trace distribution precongruence; such a result would not be true with the alternative definition of a trace distribution. ■

Prefixes

The notion of a prefix for traces can be extended to the probabilistic framework by following the same idea as for the notion of a prefix defined on probabilistic executions (cf. Section 4.2.6). A trace distribution $\mathcal{D}$ is a *prefix* of a trace distribution $\mathcal{D}'$, denoted by $\mathcal{D} \leq \mathcal{D}'$, iff for each finite trace β, $P_{\mathcal{D}}[C_\beta] \leq P_{\mathcal{D}'}[C_\beta]$. Thus, two trace distributions are equal iff each one is a prefix of the other.

Lemma 7.2.1 *Let H_1 and H_2 be two probabilistic execution fragments of a probabilistic automaton M. If $H_1 \leq H_2$, then $tdistr(H_1) \leq tdistr(H_2)$.* ■

Action Restriction

Similarly to the ordinary case, it is possible to define an action restriction operator on trace distributions. Let $\mathcal{D} = (\Omega, \mathcal{F}, P)$ be a trace distribution, and let V be a set of actions. Then the *restriction* of $\mathcal{D}$ to V, denoted by $\mathcal{D} \upharpoonright V$, is the probability space $completion((\Omega', \mathcal{F}', P'))$ where $\Omega' = \Omega \upharpoonright V$, $\mathcal{F}'$ is the σ-field generated by the sets of cones of Ω', and P' is the inverse image of P under the function that restricts traces to V.

Lemma 7.2.2 *Let $\mathcal{D}$ be a trace distribution. Then $(\mathcal{D} \upharpoonright V_1) \upharpoonright V_2 = \mathcal{D} \upharpoonright (V_1 \cap V_2)$.*

Proof. This is a direct consequence of the fact that restricting a trace to V_1 and then to V_2 is equivalent to restricting the same trace to $V_1 \cap V_2$. Formally, $\cdot \upharpoonright (V_1 \cap V_2) = (\cdot \upharpoonright V_2) \circ (\cdot \upharpoonright V_1)$. ■

Finally, we want to show that, if $M = M_1 \| M_2$, then the projection of a trace distribution of M onto M_1 and M_2 is a trace distribution of M_1 and M_2, respectively. Formally,

Proposition 7.2.3 *If $\mathcal{D} \in tdistrs(M_1 \| M_2)$, then $\mathcal{D} \upharpoonright acts(M_i) \in tdistrs(M_i)$, $i = 1, 2$.*

The converse of Proposition 7.2.3 is not true; an illustrating example is given in Section 7.4 (cf. Example 7.4.1). The rest of this section is dedicated to the proof of Proposition 7.2.3. We start with a definition of an *internal trace distribution*, which is a trace distribution that does not abstract from internal actions.

Let α be an execution of a probabilistic automaton M. The *internal trace* of α, denoted by $itrace(\alpha)$, is the subsequence of α consisting of the actions of M. Let H be a probabilistic execution fragment of M, and let f be a function from Ω_H to $\Omega = acts(H)^* \cup acts(H)^\omega$ that assigns to each execution of Ω_H its internal trace. The *internal trace distribution* of H, denoted by $itdistr(H)$, is the probability space $completion((\Omega, \mathcal{F}, P))$ where $\mathcal{F}$ is the σ-field generated by the cones of Ω, and $P = f(P_H)$. Observe that, from Proposition 3.1.4, f is a measurable function from $(\Omega_H, \mathcal{F}_H)$ to $(\Omega, \mathcal{F})$. Denote a generic internal trace distribution by $\mathcal{D}$. Denote the set of internal trace distributions of a probabilistic automaton M by $itdistrs(M)$.

Lemma 7.2.4 *Let H be a probabilistic execution fragment of a probabilistic automaton M. Then, $tdistr(H) = itdistr(H) \upharpoonright ext(H)$.*

Proof. This is a direct consequence of the fact that the set of executions of H whose trace contains a given β is the set of executions of H whose internal trace restricted to the external actions of H contains β. Formally, $\mathit{trace}(\cdot) = \mathit{itrace}(\cdot) \circ (\cdot \upharpoonright \mathit{ext}(H))$. ■

Lemma 7.2.5 *Let H be a probabilistic execution fragment of $M_1 \| M_2$, where M_1 and M_2 are two compatible probabilistic automata. Then $\mathit{itdistr}(H \lceil M_i) = \mathit{itdistr}(H) \upharpoonright \mathit{acts}(M_i)$, $i = 1, 2$.*

Proof. Let $\mathcal{P}$ denote $\mathit{itdistr}(H \lceil M_i)$, and let $\mathcal{P}'$ denote $\mathit{itdistr}(H) \upharpoonright \mathit{acts}(M_i)$. We need to show that for each finite internal trace β, $P[C_\beta] = P'[C_\beta]$. Let $\mathcal{P}''$ denote $\mathit{itdistr}(H)$. From the definition of an internal trace,

$$P[C_\beta] = P_{H \lceil M_i}[\alpha \in \Omega_{H \lceil M_i} \mid \beta \leq \mathit{itrace}(\alpha)]. \tag{7.1}$$

From the definition of $\mathcal{P}'$ and $\mathcal{P}''$,

$$P'[C_\beta] = P''[\beta' \in \Omega'' \mid \beta \leq \beta' \upharpoonright \mathit{acts}(M_i)]. \tag{7.2}$$

From the definition of $\mathit{itdistr}(H)$ and (7.2),

$$P'[C_\beta] = P_H[\alpha \in \Omega_H \mid \beta \leq \mathit{itrace}(\alpha) \upharpoonright \mathit{acts}(M_i)]. \tag{7.3}$$

Thus, from (7.1) and (7.3), we need to show that

$$P_{H \lceil M_i}[\alpha \in \Omega_{H \lceil M_i} \mid \beta \leq \mathit{itrace}(\alpha)] = P_H[\alpha \in \Omega_H \mid \beta \leq \mathit{itrace}(\alpha) \upharpoonright \mathit{acts}(M_i)]. \tag{7.4}$$

By using a characterization of the involved events as a disjoint union of cones, and by rewriting Equation 7.4 accordingly, we obtain

$$\begin{aligned} & P_{H \lceil M_i}[\bigcup_{q \in \mathit{states}(H \lceil M_i) \mid \mathit{itrace}(q) = \beta, \mathit{lact}(q) = \mathit{lact}(\beta)} C_q] \\ = \;& P_H[\bigcup_{q \in \mathit{states}(H) \mid \mathit{itrace}(q) \upharpoonright \mathit{acts}(M_i) = \beta, \mathit{lact}(q) = \mathit{lact}(\beta)} C_q]. \end{aligned} \tag{7.5}$$

Observe that for each $q \in \mathit{states}(H)$ such that $\mathit{itrace}(q) \upharpoonright \mathit{acts}(M_i) = \beta$ and $\mathit{lact}(q) = \mathit{lact}(\beta)$, the state $q \lceil M_i$ is a state of $H \lceil M_i$ such that $\mathit{itrace}(q \lceil M_i) = \beta$ and $\mathit{lact}(q \lceil M_i) = \mathit{lact}(\beta)$. Moreover, the states q of the left expression of (7.5) are partitioned by the relation that relates q and q' whenever $q \lceil M_i = q' \lceil M_i$. Thus, if we show that for each trace β and each $q \in \mathit{states}(H \lceil M_i)$ such that $\mathit{itrace}(q) = \beta$ and $\mathit{lact}(q) = \mathit{lact}(\beta)$,

$$P_{H \lceil M_i}[C_q] = P_H[\cup_{q' \in q \rceil H \mid \mathit{lact}(q') = \mathit{lact}(\beta)} C_{q'}], \tag{7.6}$$

Equation (7.5) is proved. Observe that

$$P_H[\cup_{q' \in \mathit{states}(H) \mid q' \lceil M_i = q, \mathit{lact}(q') = \mathit{lact}(\beta)} C_{q'}] = \sum_{q' \in \mathit{min}(q \rceil H)} P_H[C_{q'}], \tag{7.7}$$

since $\{q' \in \mathit{states}(H) \mid q' \lceil M_i = q, \mathit{lact}(q') = \mathit{lact}(\beta)\} = \mathit{min}(q \rceil H)$. Thus, Equation (7.6) becomes

$$P_{H \lceil M_i}[C_q] = \sum_{q' \in \mathit{min}(q \rceil H)} P_H[C_{q'}], \tag{7.8}$$

which is true from Proposition 4.3.5. ■

Lemma 7.2.6 *Let H be a probabilistic execution fragment of $M_1 \| M_2$, where M_1 and M_2 are two compatible probabilistic automata. Then $tdistr(H\lceil M_i) = tdistr(H) \upharpoonright acts(M_i)$.*

Proof. From Lemma 7.2.4,

$$tdistr(H\lceil M_i) = itdistr(H\lceil M_i) \upharpoonright ext(M_i). \tag{7.9}$$

From Lemma 7.2.5 and (7.9),

$$tdistr(H\lceil M_i) = (itdistr(H) \upharpoonright acts(M_i)) \upharpoonright ext(M_i). \tag{7.10}$$

From Lemma 7.2.2 and (7.10),

$$tdistr(H\lceil M_i) = (itdistr(H) \upharpoonright ext(H)) \upharpoonright acts(M_i). \tag{7.11}$$

From Lemma 7.2.4 and (7.11),

$$tdistr(H\lceil M_i) = tdistr(H) \upharpoonright acts(M_i), \tag{7.12}$$

which is what we needed to prove. ■

Proof of Proposition 7.2.3. Let $\mathcal{D} \in tdistrs(M_1 \| M_2)$. Then there exists a probabilistic execution H of $M_1 \| M_2$ such that $tdistr(H) = \mathcal{D}$. From Proposition 4.3.4, $H\lceil M_i$ is a probabilistic execution of M_i. From Lemma 7.2.6, $tdistr(H\lceil M_i) = \mathcal{D} \upharpoonright acts(M_i)$. Thus, $\mathcal{D} \upharpoonright acts(M_i) \in tdistrs(M_i)$. ■

7.3 Trace Distribution Preorder

Once trace distributions are defined, the trace distribution preorder can be defined as trace distribution inclusion. Formally, let M_1, M_2 be two probabilistic automata with the same external action signature. The *trace distribution preorder* is defined as follows.

$$M_1 \sqsubseteq_D M_2 \text{ iff } tdistrs(M_1) \subseteq tdistrs(M_2). \tag{7.13}$$

The trace distribution preorder is a conservative extension of the trace preorder of ordinary automata, and it preserves properties that resemble the safety properties of ordinary automata [AS85]. Here we give some examples of such properties.

Example 7.3.1 The following property is preserved by the trace distribution preorder.

> *"After some finite trace β has occurred, then the probability that some other trace β' occurs, is not greater than p."*

In fact, suppose that $M_1 \sqsubseteq_D M_2$, and suppose that M_2 satisfies the property above, while M_1 does not. Then there is a trace distribution of M_1 where the probability of β' after β conditional to β is greater than p. Since $M_1 \sqsubseteq_D M_2$, there is a trace distribution of M_2 where the probability of β' after β conditional to β is greater than p. This contradicts the hypothesis that M_2 satisfies the property above. Observe that the property above would still be preserved if we replace β' with a set of traces. ■

Example 7.3.2 The following property is preserved by the trace distribution preorder.

> "*In every computation where infinite external activity occurs with probability* 1*, if a finite trace* β *occurs, then the probability that some other trace* β' *occurs after* β *given that* β *occurs is at least* p*.*"

A more concrete instantiation of the property above is "under the hypothesis that a distributed system never deadlocks, every request of service eventually gets a response with probability at least p". This property is definitely more interesting than the property of Example 7.3.1 since it involves a progress statement, one of the property of key interest for the analysis of randomized distributed algorithms. Thus, if in a system it is always possible to avoid a deadlock, under the assumption that we always schedule a transition and under the condition that no infinite internal computation is possible, the property above guarantees progress. However, in order to be sure that if $M_1 \sqsubseteq_D M_2$ and M_2 satisfies the property above then M_1 guarantee progress, we need to make sure that from every state of M_2 it is possible to avoid deadlock and there is no possibility of infinite internal computation. Such a property must be verified separately since it is not guaranteed by the trace distribution preorder. Fortunately, there are several cases (e.g., n processes running in parallel that communicate via shared memory) where it is easy to verify that it is always possible to avoid a deadlock.

To prove that the property above is preserved, suppose that $M_1 \sqsubseteq_D M_2$, and suppose that M_2 satisfies the the property above, while M_1 does not. Then there is a trace distribution of M_1 with infinite external computation where the probability of β' after β conditional to β is greater than p. Since $M_1 \sqsubseteq_D M_2$, there is a trace distribution of M_2 with infinite external computation where the probability of β' after β conditional to β is greater than p. This contradicts the hypothesis that M_2 satisfies the property above. ■

Example 7.3.3 The following property is preserved by the trace distribution preorder.

> "*In every computation where infinite external activity occurs with probability* 1*, if a finite trace* β *occurs, then, no matter what state is reached, a trace* β' *occurs ofter* β *with probability at least* p*.*"

A more concrete instantiation of the property above is "under the hypothesis that a distributed system never deadlocks, if a process has requested a service (β), then, no matter what state is reached, either the service has received a positive acknowledgment already (β'), or a positive acknowledgment will be received eventually with probability at least p". This property is preserved by the trace distribution preorder since it is equivalent to the property of Example 7.3.2 with $p = 1$ (cf. Proposition 5.5.5 to have an idea of why this is true). ■

Essentially, the rule of thumb to determine what properties can be guaranteed to be preserved under the trace distribution preorder is the following: express the property of interest as a property ϕ of the trace distributions of a probabilistic automaton M plus a condition ψ on the structure of M. If $M_1 \sqsubseteq_D M_2$, then the trace distributions of M_1 satisfy the property ϕ. Thus, if we know that M_2 satisfies the property of interest, it is enough to verify separately that M_1 satisfies ψ in order to be guaranteed that also M_1 satisfies the property of interest.

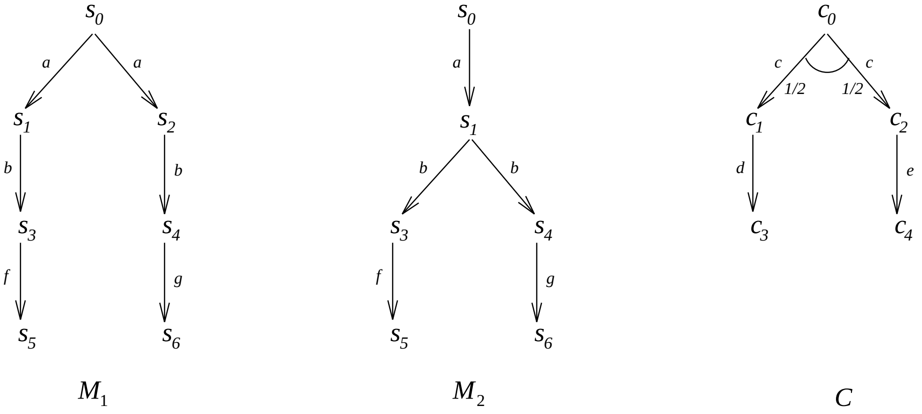


Figure 7-4: The trace distribution preorder is not a precongruence.

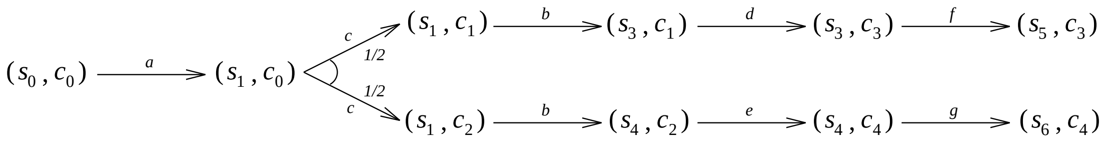


Figure 7-5: A probabilistic execution of $M_2 \| C$.

7.4 Trace Distribution Precongruence

Although the trace distribution preorder preserves some properties that are useful for the analysis of randomized distributed systems, the trace distribution preorder is not a precongruence, and thus it does not allow us to use modular analysis.

Example 7.4.1 (The trace distribution preorder is not substitutive) Consider the two probabilistic automata M_1 and M_2 of Figure 7-4. It is easy to check that M_1 and M_2 have the same trace distributions. Consider now the context C of Figure 7-4. Figure 7-5 shows a probabilistic execution of $M_2 \| C$ where there is a total correlation between the occurrence of actions d and f and actions e and g. Such a correlation cannot be obtained from $M_1 \| C$, since the choice between f and g must be resolved before knowing what action among d and e is chosen probabilistically. Thus, $M_1 \| C$ and $M_2 \| C$ do not have the same trace distributions. ■

This leads us to the definition of the *trace distribution precongruence*, denoted by $\sqsubseteq_{DC}$, as the coarsest precongruence that is contained in the trace distribution preorder. This definition of the trace distribution precongruence is not constructive, and thus it is difficult to understand what we have defined. Furthermore, we do not have any observational semantics that characterizes the trace distribution precongruence. In Section 7.5 we give an alternative characterization of the trace distribution precongruence that gives a better idea of the relation that we have defined. Here we give some examples of properties that are preserved by the trace distribution precongruence and that are not preserved by the trace distribution preorder.

Example 7.4.2 The following property is preserved by the trace distribution precongruence but not by the trace distribution preorder.

> "*After some finite trace β has occurred, no matter what state is reached, the probability that some other trace β' occurs from the state reached is not greater than p.*"

This property is not preserved by the trace distribution preorder since trace distributions cannot detect all the points where we may start to study the probability of β' to occur. However, this task is possible with the help of an external context. We use a context C that performs a fresh action o and then stops.

Suppose that $M_1 \sqsubseteq_{DC} M_2$ and suppose that M_2 satisfies the property above, while M_1 does not. Then there is a probabilistic execution H_1 of M_1 where some state q is reached after the occurrence of β, and the probability that β' occurs from q is greater than p. Consider a probabilistic execution H_1' of $M_1 \| C$ such that $H_1' \lceil M_1 = H_1$ and such that action o is scheduled exactly from the minimal state q' such that $q' \lceil M_1 = q$. Then, o occurs always after β, and the conditional probability of β' after o given that o occurred is greater than p in the trace distribution of H_1'. Since $M_1 \sqsubseteq_{DC} M_2$, then there is a probabilistic execution H_2' of $M_2 \| C$ whose trace distribution is the same as the trace distribution of H_2'. This means that there is at least one state q'' in H_2', reached immediately after the occurrence of o, where the probability that β' occurs from q'' in H_2' is greater than p. Consider $H_2' \lceil M_2$, and change its transition relation to obtain a probabilistic execution H_2 such that $H_2 \triangleright (q'' \lceil M_2) = (H_2' \lceil M_2) \triangleright (q'' \lceil M_2)$. Then the probability that β' occurs from $q'' \lceil M_2$ in H_2 is greater than p. Moreover, β has occurred when $q \lceil M_2$ is reached. This contradicts the hypothesis that M_2 satisfies the property above. ■

Example 7.4.3 The following property is preserved by the trace distribution precongruence but not by the trace distribution preorder.

> "*In every computation where infinite external activity occurs with probability* 1*, if a finite trace β occurs, then, no matter what state is reached, if another trace β'' has not occurred yet after β, then a trace β' occurs with probability at least p.*"

A more concrete instantiation of the property above is "under the hypothesis that a distributed system never deadlocks, if a process has requested a service (β) and has not received yet a refusal (β'') then, no matter what state is reached, a positive acknowledgment (β') will be received eventually with probability at least p". Observe that the main difference from the property of Example 7.3.3 is in the use of β''. The presence of β'' does not guarantee that β' occurs with probability 1.

Even in this case in the proof we use a context C with a fresh action o. Suppose that $M_1 \sqsubseteq_{DC} M_2$ and suppose that M_2 satisfies the property above, while M_1 does not. Then there is a probabilistic execution H_1 of M_1 where infinite external activity occurs such that there is a state q of H_1 that is reached after the occurrence of β and before the occurrence of β'', and such that the probability that β' occurs from q is smaller than p. Consider a probabilistic execution H_1' of $M_1 \| C$ such that $H_1' \lceil M_1 = H_1$ and such that action o is scheduled exactly from the minimal state q' such that $q' \lceil M_1 = q$. Then, o occurs always after β and before β'' occurs after β, and the conditional probability of β' after o given that o occurred is greater than p in the trace distribution of H_1'. Since $M_1 \sqsubseteq_{DC} M_2$, then there is a probabilistic execution H_2' of $M_2 \| C$ whose trace distribution is the same as the trace distribution of H_2'. This means that there is at

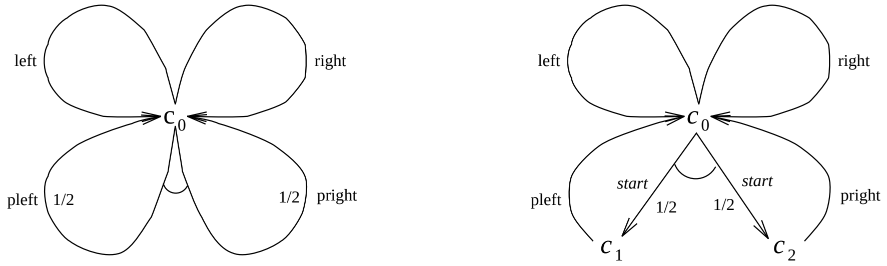


Figure 7-6: The principal context (left) and the simple principal context (right).

least one state q'' in H'_2, reached immediately after the occurrence of o, where the probability that β' occurs from q'' in H'_2 is smaller than p. Consider $H'_2\lceil M_2$, and change its transition relation to obtain a probabilistic execution H_2 such that $H_2 \rhd (q''\lceil M_2) = (H'_2\lceil M_2) \rhd (q''\lceil M_2)$. Then the probability that β' occurs from $q''\lceil M_2$ in H_2 is smaller than p. Moreover, β has occurred when $q\lceil M_2$ is reached and similarly β'' has not occurred after the occurrence of β. This contradicts the hypothesis that M_2 satisfies the property above. ■

7.5 Alternative Characterizations of the Trace Distribution Precongruence

In this section we give an alternative characterization of the trace distribution precongruence that is easier to manipulate. We define a *principal context*, denoted by C_P, and we show that there exists a context C that can distinguish two probabilistic automata M_1 and M_2 iff the principal context distinguishes M_1 and M_2.

7.5.1 The Principal Context

The principal context is a probabilistic automaton with a unique state and three self-loop transitions labeled with actions that do not appear in any other probabilistic automaton. Two self-loop transitions are deterministic (Dirac) and are labeled with action *left* and *right*, respectively; the third self-loop transition is probabilistic, where one edge leads to the occurrence of action *pleft* with probability 1/2 and the other edge leads to the occurrence of action *pright* with probability 1/2. Figure 7-6 shows the principal context.

The principal context is not a simple probabilistic automaton; however, since it does not have any action in common with any other probabilistic automaton, the parallel composition operator can be extended trivially: no synchronization is allowed. Alternatively, if we do not want a non-simple context, we can replace the principal context with the *simple principal context*, represented in Figure 7-6, as well. In this case we need to assume that also action *start* does not appear in any other probabilistic automaton. The main theorem is the following.

Theorem 7.5.1 $M_1 \sqsubseteq_{DC} M_2$ *iff* $M_1 \| C_P \sqsubseteq_D M_2 \| C_P$. ■

As a corollary we obtain an alternative characterization of the trace distribution precongruence and a compositional observational semantics for probabilistic automata. A *principal trace distri-*

bution of a probabilistic automaton M is a trace distribution of $M \| C_P$. Denote by $ptdistrs(M)$ the set $tdistrs(M \| C_P)$.

Corollary 7.5.2 *$M_1 \sqsubseteq_{DC} M_2$ iff $ptdistrs(M_1) \subseteq ptdistrs(M_2)$.* ■

The fact that the principal context is not a simple probabilistic automaton may appear to be confusing. Here we shed some light on the problem. First of all, in Chapter 4 we have defined parallel composition only for simple probabilistic automata; in this section, in order to account for the principal context, we have extended parallel composition to pairs of probabilistic automata, not necessarily simple, that do not have any action in common. This raises an immediate question: is the trace distribution precongruence defined based solely on contexts that are simple probabilistic automata or is it defined based on any compatible context according to the new extended parallel composition? The answer to this question, as it will become clear from the proof of Theorem 7.5.1, is that it does not matter because the two definitions are equivalent. That is, if there is a non-simple context that distinguishes two simple probabilistic automata M_1 and M_2, then the simple principal context distinguishes M_1 and M_2 as well.

Our choice of the principal context is just stylistic since it contains less structure than the simple principal context. The reader should keep in mind that there are infinitely many contexts with the same properties as the principal and the simple principal contexts; any one of those contexts can be chosen to give an alternative characterization to the trace distribution precongruence.

7.5.2 High Level Proof

The rest of this section is dedicated to the proof of Theorem 7.5.1. The proof is structured in several steps where at each step a generic distinguishing context C is transformed into a simpler distinguishing context C'. The proof of each transformation step is structured as follows. Given a distinguishing context C for $M_1 \sqsubseteq_D M_2$, build a simpler context C'. Suppose by contradiction that C' is not a distinguishing context and consider a trace distribution $\mathcal{D}$ of $M_1 \| C$ that is not a trace distribution of $M_2 \| C$. Let H_1 be a probabilistic execution of $M_1 \| C$ such that $tdistr(H_1) = \mathcal{D}$. Transform H_1 into a probabilistic execution H_1' of $M_1 \| C'$, and show that if there is a probabilistic execution H_2' of $M_2 \| C'$ such that $tdistr(H_2') = tdistr(H_1')$, then H_2' can be transformed into a probabilistic execution H_2 of $M_2 \| C$ such that $tdistr(H_2) = \mathcal{D}$. This leads to a contradiction.

The high level proof of Theorem 7.5.1 is then the following.

$\Longrightarrow$: Assuming that the principal context distinguishes M_1 and M_2, we show that the simple principal context distinguishes M_1 and M_2.

$\Longleftarrow$: We consider a generic context C that distinguishes M_1 and M_2, and we transform it into the principal context, showing that the principal context distinguishes M_1 and M_2. The transformation steps are the following.

1. Ensure that C does not have any action in common with M_1 and M_2 (Lemma 7.5.3);
2. Ensure that C does not have any cycles in its transition relation (Lemma 7.5.4);
3. Ensure that the branching structure of C is at most countable (Lemma 7.5.5);

4. Ensure that the branching structure of C is at most binary (Lemma 7.5.6);
5. Ensure that the probabilistic transitions of C lead to binary and uniform distributions (Lemma 7.5.7);
6. Ensure that each action of C is external and appears exactly in one edge of the transition relation of C (Lemma 7.5.8);
7. Ensure that each state of C enables two deterministic transitions and one probabilistic transition with a uniform binary distribution (Lemma 7.5.9);
8. Rename all the actions of the context of 7 according to the action names of the principal context and then collapse all the states of the new context into a unique state, leading to the principal context (Lemma 7.5.10).

7.5.3 Detailed Proof

Lemma 7.5.3 *Let C be a distinguishing context for two probabilistic automata M_1 and M_2. Then there exists a distinguishing context C' for M_1 and M_2 with no actions in common with M_1 and M_2. C' is called a* separated context.

Proof. The context C' is built from C be replacing each action a in common with M_1 and M_2, called a *shared* action, with two new actions a_1, a_2, and by replacing each transition $(c, a, \mathcal{P})$ of C with two transitions (c, a_1, c') and $(c', a_2, \mathcal{P})$, where c' denotes a new state that is used only for the transition $(c, a, \mathcal{P})$. We denote c' also by $c_{(c,a,\mathcal{P})}$ when convenient. We also denote the set of actions of the kind a_1 and a_2 by V_1 and V_2, respectively.

Let $\mathcal{D}$ be a trace distribution of $M_1 \| C$ that is not a trace distribution of $M_2 \| C$. Consider a probabilistic execution H_1 of $M_1 \| C$ such that $\mathit{tdistr}(H_1) = \mathcal{D}$, and consider the scheduler that leads to H_1. Apply to $M_1 \| C'$ the same scheduler with the following modification: whenever a transition $((s_1, c), a, \mathcal{P}_1 \otimes \mathcal{P})$ is scheduled in $M_1 \| C$, schedule $((s_1, c), a_1, \mathcal{D}((s_1, c')))$, where c' is $c_{(c,a,\mathcal{P})}$, followed by $((s_1, c'), a, \mathcal{P}_1 \otimes \mathcal{D}(c'))$, and, for each $s_1' \in \Omega_1$, followed by $((s_1', c'), a_2, \mathcal{D}(s_1') \otimes \mathcal{P})$. Denote the resulting probabilistic execution by H_1' and the resulting trace distribution by $\mathcal{D}'$. Then,

$$\mathcal{D}' \upharpoonright \mathit{acts}(M_1 \| C) = \mathcal{D}. \tag{7.14}$$

To prove (7.14) we define a new construction, called *collapse* and abbreviated with *clp*, to be applied to probabilistic executions of $M_i \| C'$, $i = 1, 2$, where each occurrence of a shared action a is followed immediately by an occurrence of its corresponding action a_2.

Let H' be a probabilistic execution of $M_i \| C'$ where each occurrence of a shared action a is followed immediately by an occurrence of its corresponding action a_2. For convenience denote $\mathit{clp}(H')$ by H. A state q of H' is *closed* if each occurrence of a shared action a is followed eventually by an occurrence of the corresponding action a_2. For each closed state q of H', let $\mathit{clp}(q)$ be obtained from q as follows: each sequence

$$(s_0, c_0) a_1 (s_0, c_{tr}) \tau_2 (s_2, c_{tr}) \cdots \tau_k (s_k, c_{tr}) a (s, c_{tr}) a_2 (s, c)$$

is replaced with

$$(s_0, c_0) \tau_2 (s_2, c_0) \cdots \tau_k (s_k, c_0) a (s, c),$$

and each sequence

$$(s_0, c_0)a_1(s_1, c_{tr})\tau_2(s_2, c_{tr}) \cdots \tau_k(s_k, c_{tr})$$

occurring at the end of q is replaced with

$$(s_0, c_0)\tau_2(s_2, c_0) \cdots \tau_k(s_k, c_0).$$

Define

$$states(H) \triangleq \{clp(q) \mid q \in states(H'), closed(q)\}. \tag{7.15}$$

Let $(q, \mathcal{P})$ be a restricted transition of H' where q is a closed state, and suppose that no action of $V_1 \cup V_2$ occurs. Consider a pair (a, q') of Ω. If a is not a shared action, then let

$$\mathcal{P}_{(a,q')} \triangleq \mathcal{D}((a, clp(q'))); \tag{7.16}$$

if a is a shared action, then let

$$\Omega_{(a,q')} \triangleq \{(a, clp(q'')) \mid (a_2, q'') \in \Omega_{q'}^{H'}\}, \tag{7.17}$$

and for each $(a, q''') \in \Omega_{(a,q')}$, let

$$P_{(a,q')}[(a, q''')] \triangleq P_{q'}[a_2 \times clp^{-1}(q''')], \tag{7.18}$$

where for each state q of H, $clp^{-1}(q)$ is the set of closed states q' of H' such that $clp(q') = q$. The transition $clp((q, \mathcal{P}))$ is defined to be

$$clp((q, \mathcal{P})) \triangleq \left(clp(q), \sum_{(a,q') \in \Omega} P[(a, q')]\mathcal{P}_{(a,q')}\right). \tag{7.19}$$

For the transition relation of H, consider a state q of H Let $min(clp^{-1}(q))$ be the set of minimal states of $clp^{-1}(q)$ under prefix ordering. For each state $\bar{q} \in clp^{-1}(q)$, let

$$\bar{p}_{\bar{q}}^{clp^{-1}(q)} \triangleq \frac{P_{H'}[C_{\bar{q}}]}{\sum_{q' \in min(clp^{-1}(q))} P_{H'}[C_{q'}]}. \tag{7.20}$$

The transition enabled in H from q is

$$\sum_{q' \in clp^{-1}(q)} \bar{p}_{q'}^{clp^{-1}(q)} P_{q'}^{H'}[acts(M_i \| C)] clp(tr_{q'}^{H'} \restriction acts(M_i \| C)). \tag{7.21}$$

Note the similarity with the definition of the projection of a probabilistic execution fragment (cf. Section 4.3.2).

The probabilistic execution H satisfies the following properties.

a. H is a probabilistic execution of $M_i \| C$.

   The fact that each state of H is reachable can be shown by a simple inductive argument; the fact that each state of H is a finite execution fragment of $M_i \| C$ follows from a simple analysis of the definition of clp.

   From (7.21) it is enough to check that for each closed state q' of H', the transition $clp(tr_{q'}^{H'} \restriction acts(M_i \| C))$ is generated by a combination of transitions of $M_i \| C$. Since $tr_{q'}^{H'}$ is a transition of H', $(tr_{q'}^{H'} \restriction acts(M_i \| C))$ can be expressed as $\sum_j p_j(q' \frown tr_j)$, where each tr_j is a transition of $M_i \| C'$. We distinguish three cases.

1. tr_j is a non-shared transition of M_i.

   Then $tr_j = ((s,c), a, \mathcal{P} \otimes \mathcal{D}(c))$ for some action a and probability space $\mathcal{P}$, where $(s,c) = \mathit{lstate}(q')$. Let $\mathit{lstate}(\mathit{clp}(q')) = (s', c')$. Then, $s' = s$, as it follows directly from the definition of clp. Define tr'_j to be the transition $((s,c'), a, \mathcal{P} \otimes \mathcal{D}(c'))$. Then tr'_j is a transition of $M_i \| C$ and $\mathit{clp}(q' \frown tr_j) = \mathit{clp}(q') \frown tr'_j$

2. tr_j is a non-shared transition of C'.

   Then $tr_j = ((s,c), a, \mathcal{D}(s) \otimes \mathcal{P})$ for some action a and probability space $\mathcal{P}$, where $(s,c) = \mathit{lstate}(q')$. Let $\mathit{lstate}(\mathit{clp}(q')) = (s', c')$. Then, $s' = s$ and $c' = c$, as it follows directly from the definition of clp after observing that q' must be a closed state in order to enable tr_j. Define tr'_j to be tr_j. Then tr'_j is a transition of $M_i \| C$ and $\mathit{clp}(q' \frown tr_j) = \mathit{clp}(q') \frown tr'_j$

3. tr_j is a shared transition.

   Then $tr_j = ((s, c_{tr}), a, \mathcal{P} \otimes \mathcal{D}(c_{tr}))$ for some action a and probability space $\mathcal{P}$, where $(s, c_{tr}) = \mathit{lstate}(q')$. In particular, c_{tr} is one of the states that are added to those of C, and tr is a simple transition of C with action a. Moreover, from each state $(s', c_{tr}) \in \Omega_{\mathcal{P} \otimes \mathcal{D}(c_{tr})}$, there is a transition $((s', c_{tr}), a_2, \mathcal{D}(s') \otimes \mathcal{P}_{tr})$ enabled. Let $\mathit{lstate}(\mathit{clp}(q')) = (s', c')$. Then, $s' = s$. Define tr'_j to be $((s, c'), a, \mathcal{P} \otimes \mathcal{P}_{tr})$. Then, from the definition of C', tr'_j is a transition of $M_i \| C$.

Observe that clp distributes over combination of transitions. Moreover, from Equation (7.19), observe that for each j $\mathit{clp}(q' \frown tr_j) = \mathit{clp}(q') \frown tr'_j$. Thus, $\mathit{clp}(tr_{q'}^{H'} \restriction \mathit{acts}(M_i \| C)) = \mathit{clp}(q') \frown (\sum_j p_j tr'_j)$, which is generated by a combination of transitions of $M_i \| C$.

b. For each state q of H,

$$P_H[C_q] = \sum_{q' \in \mathit{min}(\mathit{clp}^{-1}(q))} P_{H'}[C_{q'}]. \tag{7.22}$$

This is shown by induction on the length of q. If q consists of a start state only, then the result is trivial. Otherwise, from the definition of the probability of a cone, Equation (7.21), and a simple algebraic simplification,

$$P_H[C_{qas}] = P_H[C_q] \left(\sum_{q' \in \mathit{clp}^{-1}(q)} \bar{p}_{q'}^{\mathit{clp}^{-1}(q)} F_{q'}(qas) \right), \tag{7.23}$$

where $F_{q'}(qas)$ expresses the probability of the completions of q' to a state whose collapse gives qas without using actions from $V_1 \cup V_2$ in the first transition. Formally, if a is not a shared action, then $F_{q'}(qas)$ is $P_{q'}^{H'}[a \times \mathit{clp}^{-1}(qas)]$; otherwise, $F_{q'}(qas)$ is $P_{q'}^{H'}[(a, q'a(s', c_{tr}))] P_{q'a(s', c_{tr})}^{H'}[(a_2, q'a(s', c_{tr})a_2(s', c))]$, where $c_{tr} = \mathit{lstate}(q') \lceil C'$, and $s = (s', c)$. In the first case, $\Omega_{q'}^{H'} \cap (\{a\} \times \mathit{clp}^{-1}(qas))$ contains only one element, say $(a, q'as'')$, and $P_{H'}[C_{q'}] F_{q'}(qas)$ gives $P_{H'}[C_{q'as''}]$; in the second case $P_{H'}[C_{q'}] F_{q'}(qas)$ gives $P_{H'}[C_{(q'a(s', c_{tr})a_2 s)}]$.

Observe that the states of $min(clp^{-1}(qas))$ are the states of the form described above (simple cases analysis). Thus, by applying induction to (7.23), using (7.20), simplifying algebraically, and using the observations above,

$$P_H[C_{qas}] = \sum_{q' \in min(clp^{-1}(qas))} P_{H'}[C_{q'}]. \tag{7.24}$$

c. $tdistr(H) = tdistr(H') \upharpoonright acts(M_i \| C)$.

Let β be a finite trace of H or H'. Then $\{\alpha \in \Omega_{H'} \mid \beta \leq trace(\alpha) \upharpoonright acts(M_i \| C)\}$ can be expressed as a union of disjoint cones $\cup_{q \in \Theta} C_q$ where, if the last action of β is a and a is not a shared action,

$$\Theta = \{q \in states(H') \mid trace(q) \upharpoonright acts(M_i \| C) = \beta, lact(q) = a\}, \tag{7.25}$$

and if the last action of β is a and a is a shared action,

$$\Theta = \{q \in states(H') \mid trace(q) \upharpoonright acts(M_i \| C) = \beta, lact(q) = a_2\}. \tag{7.26}$$

Observe that Θ is a set of closed states. The set $clp(\Theta)$ is the set

$$clp(\Theta) = \{q \in states(H) \mid trace(q) = \beta, lact(q) = a\}, \tag{7.27}$$

which is a characterization of $\{\alpha \in \Omega_H \mid \beta \leq trace(\alpha)\}$ as a union of disjoint cones. Observe that $min(clp^{-1}(clp(\Theta))) = \Theta$. Moreover, for each $q_1 \neq q_2$ of $clp(\Theta)$, $clp^{-1}(q_1) \cap clp^{-1}(q_2) = \emptyset$. Thus, from (7.22), $P_{H'}[\cup_{q \in \Theta} C_q] = P_H[\cup_{q \in clp(\Theta)} C_q]$. This is enough to conclude.

To complete the proof of (7.14) it is enough to observe that $H_1 = clp(H_1')$. Property (7.14) is then expressed by property (c).

Suppose by contradiction that it is possible to obtain $\mathcal{D}'$ from $M_2 \| C'$. Consider the scheduler that leads to $\mathcal{D}'$ in $M_2 \| C'$, and let H_2' be the corresponding probabilistic execution. First, we build a new probabilistic execution H_2'' of $M_2 \| C'$ whose trace distribution is $\mathcal{D}'$, and such that each shared action a is followed immediately by its corresponding action a_2. Then we let H_2 be $clp(H_2'')$. This leads to a contradiction since $tdistr(H_2) = \mathcal{D}$. The rest of the proof is dedicated to the construction of H_2''.

For each state q of H_2', let $exch(q)$ be the set of sequences q' that can be obtained from q as follows: each sequence

$$(s_0, c_{tr}) a (s_1, c_{tr}) \tau_2 (s_2, c_{tr}) \cdots \tau_h (s_h, c_{tr}) a_2 (s_h, c)$$

is replaced with

$$(s_0, c_{tr}) a (s_1, c_{tr}) a_2 (s_1, c) \tau_2 (s_2, c) \cdots \tau_h (s_h, c),$$

each sequence

$$(s_0, c_{tr}) a (s_1, c_{tr}) \tau_2 (s_2, c_{tr}) \cdots \tau_h (s_h, c_{tr})$$

occurring at the end of q is replaced with

$$(s_0, c_{tr})a(s_1, c_{tr})a_2(s_1, c)\tau_2(s_2, c)\cdots\tau_h(s_h, c),$$

where c is any of the states that a_2 may lead to from c_{tr}, and each sequence

$$(s_0, c_{tr})a(s_1, c_{tr})$$

occurring at the end of q, where a is a shared action, either it is replaced with

$$(s_0, c_{tr})a(s_1, c_{tr})a_2(s_1, c),$$

where c is any of the states that a_2 may lead to from c_{tr}, or it is not replaced. Then, define

$$states(H_2'') \triangleq \bigcup_{q \in states(H_2')} exch(q). \tag{7.28}$$

Let $(q, \mathcal{P})$ be a restricted transition of H_2', and suppose that no action of V_2 occurs. Let q' be a state of $exch(q)$ that does not end with a shared action. Then, for each $(a, q_1) \in \Omega$ there is exactly one $q_1' \in exch(q_1)$ such that $q' \leq q_1'$ and $|q_1'| = |q'| + 1$ (simple analysis of the definition of $exch$). Denote such q_1' by $exch_{q'}(q_1)$. Let $\Omega' = \{(a, exch_{q'}(q_1) \mid (a, q_1) \in \Omega\}$, and let, for each $(a, q_1') \in \Omega'$, $P'[(a, q_1')] = P[(a \times exch^{-1}(q_1'))]$, where $exch^{-1}(q)$ is the set of states q' of H_2' such that $q \in exch(q')$. Then define the transition $exch_{q'}((q, \mathcal{P}))$ to be

$$exch_{q'}((q, \mathcal{P})) \triangleq (q', \mathcal{P}'). \tag{7.29}$$

For each state q of H_2'', let $min(exch^{-1}(q))$ be the set of minimal states of $exch^{-1}(q)$ under prefix ordering. For each state q' of $exch^{-1}(q)$, where q is closed, let

- $p_{q'}^{q} \triangleq P_{H_2'}[C_{q'}]$ if q' is *closed*, i.e., if each occurrence of a shared action a is followed eventually by an occurrence of its corresponding action a_2;
- $p_{q'}^{q} \triangleq P_{H_2'}[C_{q'}]P_{tr}[c]$ if q' is open, where $lstate(q')\lceil C' = c_{tr}$ and $lstate(q)\lceil C = c$.

For each $q' \in exch^{-1}(q)$, let

$$\bar{p}_{q'}^{exch^{-1}(q)} \triangleq \frac{p_{q'}^{q}}{\sum_{q'' \in min(exch^{-1}(q))} p_{q''}^{q}}. \tag{7.30}$$

If the last action of q is a shared action a, and $lstate(q) = (s, c_{tr})$, then the transition enabled from q in H_2'' is

$$q \frown ((s, c_{tr}), a_2, \mathcal{D}(s) \otimes \mathcal{P}_{tr}). \tag{7.31}$$

If the last action of q is not a shared action, then the transition enabled from q in H_2'' is

$$\sum_{q' \in exch^{-1}(q)} \bar{p}_{q'}^{exch^{-1}(q)} P_{q'}^{H_2'}[acts(H_2')\backslash V_2] exch_q(tr_{q'}^{H_2'} \restriction (acts(H_2')\backslash V_2)). \tag{7.32}$$

The probabilistic execution H_2' satisfies the following properties.

a. H_2'' is a probabilistic execution of $M_2 \| C'$.

The fact that each state of H_2'' is reachable can be shown by a simple inductive argument; the fact that each state of H_2'' is a finite execution fragment of $M_2 \| C'$ follows from a simple analysis of the definition of *exch*.

We need to check that for each state q of H_2'' the transition enabled from q in H_2'' is generated by a combination of transitions of $M_2 \| C'$. If the last action of q is a shared action, then the result follows immediately from Expression (7.31) and the definition of C'. If the last action of q is not a shared action, then consider a state $q' \in exch^{-1}(q)$. The transition $tr_{q'}^{H_2'} \restriction (acts(H_2') \backslash V_2)$ can be expressed as $\sum_i p_i (q' \frown tr_i)$, where each tr_i is a transition of $M_2 \| C'$ enabled from $lstate(q')$. We distinguish three cases.

1. tr_i is a non-shared transition of M_2.
   Then $tr_i = ((s,c), a, \mathcal{P} \otimes \mathcal{D}(c))$ for some action a and probability space $\mathcal{P}$, where $(s,c) = lstate(q')$. Let $lstate(q) = (s', c')$. Then, $s' = s$. Define tr_i' to be the transition $((s,c'), a, \mathcal{P} \otimes \mathcal{D}(c'))$. Then tr_i' is a transition of $M_2 \| C'$ and $exch_q(q' \frown tr_i) = q \frown tr_i'$.
2. tr_i is a non-shared transition of C'.
   Then $tr_i = ((s,c), a, \mathcal{D}(s) \otimes \mathcal{P})$ for some action a and probability space $\mathcal{P}$, where $(s,c) = lstate(q')$. Let $lstate(q) = (s', c')$. Then, $s' = s$ and $c = c'$. Define tr_i' to be tr_i. Then tr_i' is a transition of $M_2 \| C'$ and $exch_q(q' \frown tr_i) = q \frown tr_i'$.
3. tr_i is a shared transition.
   Then $tr_i = ((s,c), a, \mathcal{P} \otimes \mathcal{D}(c))$ for some action a and probability space $\mathcal{P}$, where $(s,c) = lstate(q')$. Let $lstate(q) = (s', c')$. Then, $s' = s$ and $c = c'$. Define tr_i' to be tr_i. Then tr_i' is a transition of $M_2 \| C'$ and $exch_q(q' \frown tr_i) = q \frown tr_i'$.

Observe that *exch* distributes over combination of transitions. Thus, $exch_q((tr_{q'}) \restriction (acts(H_2') \backslash V_2))$ can be expressed as $\sum_i p_i (q \frown tr_i')$, which is generated by a combination of transitions of $M_2 \| C'$. From (7.32), the transition enabled from q in H_2'' is generated by a combination of transitions of $M_2 \| C'$.

b. For each state q of H_2'',

$$P_{H_2''}[C_q] = \begin{cases} \sum_{q' \in min(exch^{-1}(q))} P_{H_2'}[C_{q'}] & \text{if } q \text{ ends with a shared action,} \\ \sum_{q' \in min(exch^{-1}(q))} p_{q'}^{q} & \text{otherwise.} \end{cases} \tag{7.33}$$

The proof is by induction on the length of q. If q consists of a start state only, then the result is trivial. Otherwise, consider $P_{H_2''}[C_{qas}]$. We distinguish two cases.

1. q is open.
   In this case, since in H_2' each shared action is followed immediately by the corresponding action of V_2, a is an action of V_2. Moreover, from the definition of *exch*,
   $$exch^{-1}(q) = min(exch^{-1}(qas)) = min(exch^{-1}(q)), \tag{7.34}$$
   and all the elements of $exch^{-1}(q)$ are open states. From induction,
   $$P_{H_2''}[C_q] = \sum_{q' \in min(exch^{-1}(q))} P_{H_2'}[C_{q'}]. \tag{7.35}$$

Let $c = s\lceil M_2$, and let $c_{tr} = \mathit{lstate}(q)\lceil C'$. Then, for each $q' \in \mathit{min}(\mathit{exch}^{-1}(q))$, $c_{tr} = \mathit{lstate}(q')\lceil C'$, and

$$p_{q'}^{qas} = P_{H'_2}[C_{q'}]P_{tr}[c]. \tag{7.36}$$

Moreover, $P_q^{H''_2}[(a, qas)] = P_{tr}[c]$. Thus, from the definition of the probability of a cone and (7.35),

$$P_{H''_2}[C_{qas}] = \sum_{q' \in \mathit{min}(\mathit{exch}^{-1}(q))} P_{H'_2}[C_{q'}]P_{tr}[c]. \tag{7.37}$$

By using the fact that $\mathit{min}(\mathit{exch}^{-1}(q)) = \mathit{min}(\mathit{exch}^{-1}(qas))$, and using (7.36), we obtain

$$P_{H''_2}[C_{qas}] = \sum_{q' \in \mathit{min}(\mathit{exch}^{-1}(qas))} p_{q'}^{qas}. \tag{7.38}$$

2. q is closed.

In this case, from the definition of the probability of a cone and (7.32),

$$P_{H''_2}[C_{qas}] = P_{H''_2}[C_q]\left(\sum_{q' \in \mathit{exch}^{-1}(q)} \bar{p}_{q'}^{\mathit{exch}^{-1}(q)} P_{q'}^{H'_2}[a \times \mathit{exch}^{-1}(qas)]\right). \tag{7.39}$$

Let $Ptr_q[q']$ denote $P_{tr}[c]$, where $c = \mathit{lstate}(q)\lceil C'$, and $c_{tr} = \mathit{lstate}(q')\lceil C'$. Then, from induction and (7.30),

$$\begin{aligned} P_{H''_2}[C_{qas}] \;=\; & \sum_{q' \in \mathit{exch}^{-1}(q) | \mathit{closed}(q')} P_{H'_2}[C_{q'}]P_{q'}^{H'_2}[a \times \mathit{exch}^{-1}(qas)] + \qquad (7.40) \\ & \sum_{q' \in \mathit{exch}^{-1}(q) | \mathit{open}(q')} P_{H'_2}[C_{q'}]Ptr_q[q']P_{q'}^{H'_2}[a \times \mathit{exch}^{-1}(qas)]. \end{aligned}$$

We distinguish two subcases.

(a) a is a shared action.

In this case each state q' of $\mathit{exch}^{-1}(q)$ such that $P_{q'}^{H'_2}[a \times \mathit{exch}^{-1}(qas)] > 0$ is closed. Thus, only the first summand of (7.40) is used. Moreover, each state of $\mathit{min}(\mathit{exch}^{-1}(qas))$ is captured by Expression (7.40). Thus, $P_{H'_2}[C_{qas}] = \sum_{q' \in \mathit{min}(\mathit{exch}^{-1}(qas))} P_{H'_2}[C_{q'}]$. Observe that qas is open.

(b) a is not a shared action.

In this case, for each $q' \in \mathit{exch}^{-1}(q)$, if q' is closed, then all the states reached in $\Omega_{q'} \cap (\{a\} \times \mathit{exch}^{-1}(qas))$ are closed, and if q' is open, then all the states reached in $\Omega_{q'} \cap (\{a\} \times \mathit{exch}^{-1}(qas))$ are open. Moreover, each state of $\mathit{min}(\mathit{exch}^{-1}(qas))$ is captured by Expression (7.40). Thus, from the definition of $p_{q'}^{qas}$, $P_{H'_2}[C_{qas}] = \sum_{q' \in \mathit{min}(\mathit{exch}^{-1}(qas))} p_{q'}^{qas}$. Observe that qas is closed.

c. $\mathit{tdistr}(H'_2) = \mathit{tdistr}(H''_2)$.

Let β be a finite trace of H'_2 or H''_2. Then $\{\alpha \in \Omega_{H'_2} \mid \beta \leq \mathit{trace}(\alpha)\}$ can be expressed as a union of disjoint cones $\cup_{q \in \Theta} C_q$ where

$$\Theta = \{q \in \mathit{states}(H') \mid \mathit{trace}(q) = \beta, \mathit{lact}(q) = \mathit{lact}(\beta)\}. \tag{7.41}$$

We distinguish two cases.

1. β does not end with an action of V_2.

   The set $\Theta' = \{q \in \mathit{exch}(\Theta) \mid \mathit{lact}(q) = \mathit{lact}(\beta)\}$ is a characterization of $\{\alpha \in \Omega_{H_2''} \mid \beta \leq \mathit{trace}(\alpha)\}$ as a union of disjoint cones. Observe that $\mathit{min}(\mathit{exch}^{-1}(\Theta')) = \Theta$ and that for each pair of states $q_1 \neq q_2$ of Θ', $\mathit{min}(\mathit{exch}^{-1}(q_1)) \cap \mathit{min}(\mathit{exch}^{-1}(q_2)) = \emptyset$. Thus, if β ends with a shared action, then (7.33) is sufficient to conclude that $P_{H_2'}[\{\alpha \in \Omega_{H_2'} \mid \beta \leq \mathit{trace}(\alpha)\}] = P_{H_2''}[\{\alpha \in \Omega_{H_2''} \mid \beta \leq \mathit{trace}(\alpha)\}]$; if β does not end with a shared action, then, since all the states of Θ are closed, Equation (7.33) together with the definition of $p_{q'}^{q}$ are sufficient to conclude.

2. β ends with an action of V_2.

   In this case $\beta = \beta' a_2$ for some action $a_2 \in V_2$. Observe that, both in H_2' and H_2'', after the occurrence of a shared action a the corresponding action a_2 occurs with probability 1: for H_2' recall that $\mathit{tdistr}(H_2') \upharpoonright \mathit{acts}(M_2 \| C) = \mathcal{D}$; for H_2'' see (7.31). Thus, the probability of β is the same as the probability of β', and the problem is reduced to Case 1. ■

Lemma 7.5.4 *Let C be a distinguishing separated context for two probabilistic automata M_1 and M_2. Then there exists a distinguishing cycle-free separated context C' for M_1 and M_2.*

Proof. C' can be built by unfolding C. Every scheduler for $M_i \| C$ can be transformed into a scheduler for $M_i \| C'$ and vice versa, leading to the same trace distributions. ■

Lemma 7.5.5 *Let C be a distinguishing cycle-free, separated context for two probabilistic automata M_1 and M_2. Then there exists a distinguishing cycle-free separated context C' for M_1 and M_2 with a transition relation that is at most countably branching.*

Proof. Let $\mathcal{D}$ be a trace distribution of $M_1 \| C$ that is not a trace distribution of $M_2 \| C$. Consider the corresponding probabilistic execution H. Observe that H has at most countably many states, and that at each state of H there are at most countably many transitions of C that are scheduled. Thus, in total, only countably many transitions of C are used to generate $\mathcal{D}$. Then C' is C without the unused transitions. ■

Lemma 7.5.6 *Let C be a distinguishing cycle-free, separated context for two probabilistic automata M_1 and M_2 such that the transition relation of C is at most countably branching. Then there exists a distinguishing cycle-free separated context C' for M_1 and M_2 that at each state either enables two deterministic transitions or a unique probabilistic transition with two possible outcomes. C' is called a* binary separated context*.*

Proof. For each state s of C, choose a new action start_s. Let s enable the transitions $\mathit{tr}_1, \mathit{tr}_2, \ldots$, where each tr_i is a transition $(s, a_i, \mathcal{P}_i)$. The transition relation of C' is obtained in two phases. First, a transition is chosen nondeterministically as shown in Figure 7-7, where each symbol $\bullet$ denotes a distinct state and each symbol τ denotes a distinct internal action; then, for each state $\bullet_i$, the transition tr_i is encoded as follows. Let Ω_i be $\{s_{i,1}, s_{i,2}, \ldots\}$, $p_{i,j} \triangleq P_i[s_{i,j}]$, and $\bar{p}_{i,j} \triangleq \sum_{k \geq j} p_{i,k}$. The transition relation from $\bullet_i$ is represented in Figure 7-8, where each

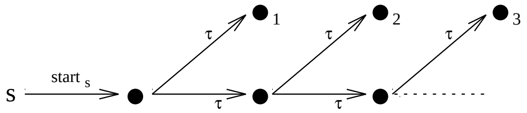


Figure 7-7: Nondeterministic choice of a transition.

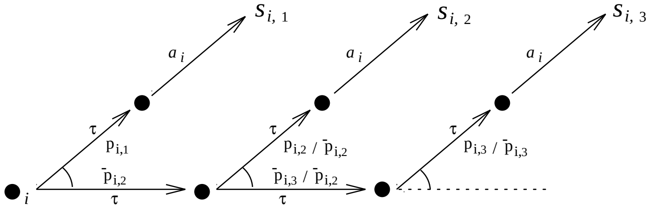


Figure 7-8: Transforming a transition into binary transitions.

symbol $\bullet$ denotes a distinct state and each symbol τ denotes a distinct internal action. Observe that by scheduling all the transitions of the diagram above, for each j we have

$$P[s_{i,j}] = P_i[s_{i,j}], \tag{7.42}$$

where $P[s_{i,j}]$ is the probability of reaching $s_{i,j}$ from $\bullet_i$. Denote the set of actions of the kind $start_s$ by V_{start}. Denote the auxiliary actions of C' that occur between a *start* action and a state $\bullet_j$ by V_1, and denote the auxiliary actions of C' that occur between a state $\bullet_j$ and the corresponding occurrence of action a_j by V_2.

Let $\mathcal{D}$ be a trace distribution of $M_1\|C$ that is not a trace distribution of $M_2\|C$. Consider a probabilistic execution H_1 of $M_1\|C$ whose trace distribution is $\mathcal{D}$ in $M_1\|C$, and consider the scheduler that leads to H_1 in $M_1\|C$. Apply to $M_1\|C'$ the same scheduler with the following modification: whenever some transition of C is scheduled, schedule the *start* action from C', then schedule the internal transitions to choose the transition of C to perform with the right probability, and then schedule the transitions of the chosen transition till the corresponding external action of C occurs. Denote the resulting probabilistic execution by H_1' and the resulting trace distribution by $\mathcal{D}'$. Then,

$$\mathcal{D}' \upharpoonright acts(M_1\|C) = \mathcal{D}. \tag{7.43}$$

To prove (7.43), we define a new construction, called *shrink* and abbreviated with *shr*, to be applied to probabilistic executions of $M_i\|C'$ such that no action of M_i occurs between a state of the form $\bullet_j$ and the occurrence of the corresponding action a_j of C, and such that all the transitions between a state of the kind $\bullet_j$ and the corresponding occurrences of action a_j are scheduled.

Let H' be such a probabilistic execution of $M_i\|C'$. Denote $shr(H')$ by H. A state q of H' is *closed* if each occurrence of a state of the kind $\bullet_j$ is followed eventually by the occurrence of the corresponding action a_j. For each state q of H' let $shr(q)$ be obtained from q as follows: each sequence

$$(s_0, c_0) start_{c_0} (s_0, \bullet) b_1 (s_1, \bullet) \cdots b_h (s_h, \bullet_j) \tau_1 (s_h, \bullet) \cdots \tau_k (s_h, \bullet) a_j (s, c)$$

is replaced with

$$(s_0, c_0) b_{i_1} (s_{i_1}, c_0) \cdots b_{i_l} (s_{i,l}, c_0) a_j (s, c),$$

where $i_1, \ldots, i_l$ is the ordered sequence of the indexes of the b's that are actions of M_i, and each sequence either of the form

$$(s_0, c_0) start_{c_0} (s_0, \bullet) b_1 (s_1, \bullet) \cdots b_h (s_h, \bullet_j) \tau_1 (s_h, \bullet) \cdots \tau_k (s_h, \bullet)$$

or of the form

$$(s_0, c_0) start_{c_0} (s_0, \bullet) b_1 (s_1, \bullet) \cdots b_h (s_h, \bullet)$$

occurring at the end of q is replaced with

$$(s_0, c_0) b_{i_1} (s_{i_1}, c_0) \cdots b_{i_l} (s_{i,l}, c_0),$$

where $i_1, \ldots, i_l$ is the ordered sequence of the indexes of the b's that are actions of M_i. Then,

$$states(H) \triangleq \{shr(q) \mid q \in states(H')\}. \tag{7.44}$$

Let $(q, \mathcal{P})$ be a restricted transition of H', and suppose that no action of $acts(C') \backslash acts(C)$ occurs. Let $\Omega' = \{(a, shr(q')) \mid (a, q') \in \Omega\}$, and for each $(a, q'') \in \Omega'$, let $P'[(a, q'')] = P[a \times shr^{-1}(q'')]$, where $shr^{-1}(q)$ is the set of states q' of H' such that $shr(q') = q$. Then the transition $shr((q, \mathcal{P}))$ is defined to be

$$shr((q, \mathcal{P})) \triangleq (shr(q), \mathcal{P}). \tag{7.45}$$

For the transition relation of H, consider a state q of H, and let $min(shr^{-1}(q))$ be the set of minimal states of $shr^{-1}(q)$ under prefix ordering. For each state $\bar{q} \in shr^{-1}(q)$, let

$$\bar{p}_{\bar{q}}^{shr^{-1}(q)} \triangleq \frac{P_{H'}[C_{\bar{q}}]}{\sum_{q' \in min(shr^{-1}(q))} P_{H'}[C_{q'}]}. \tag{7.46}$$

The transition enabled from q in H is

$$\sum_{q' \in shr^{-1}(q)} \bar{p}_{\bar{q}}^{shr^{-1}(q)} P_{q'}^{H'}[acts(M_i \| C)] shr(tr_{q'}^{H'} \restriction acts(M_i \| C)). \tag{7.47}$$

The probabilistic execution H satisfies the following properties.

a. H is a probabilistic execution of $M_i \| C$.

   The fact that each state of H is reachable can be shown by a simple inductive argument; the fact that each state of H is a finite execution fragment of $M_i \| C$ follows from a simple analysis of the definition of shr.

   We need to show that for each state q of H the transition of Expression (7.47) is generated by a combination of transitions of $M_i \| C$. The states of $shr^{-1}(q)$ that enable some action of $M_i \| C$ can be partitioned into two sets Θ_c and Θ_o of closed and open states, respectively.

   We analyze Θ_c first. Let $q' \in \Theta_c$. Since $tr_{q'}$ is a transition of H', $(tr_{q'} \restriction acts(M_i \| C))$ can be expressed as $\sum_j p_j (q' \frown tr_j)$, where each tr_j is a transition of $M_i \| C'$. We distinguish two cases.

1. tr_j is a transition of M_i.
   Then $tr_j = ((s,c), a, \mathcal{P} \otimes \mathcal{D}(c))$ for some action a and probability space $\mathcal{P}$, where $(s,c) = \mathit{lstate}(q')$. Let $\mathit{lstate}(\mathit{shr}(q')) = (s', c')$. Then, $s' = s$, as it follows directly from the definition of shr. Moreover, $(s, a, \mathcal{P})$ is a transition of M_i. Define tr'_j to be the transition $((s,c'), a, \mathcal{P} \otimes \mathcal{D}(c'))$. Then tr'_j is a transition of $M_i \| C$ and $\mathit{shr}_q(q' \frown tr_i) = q \frown tr'_j$.
2. tr_j is a transition of C'.
   This case is not possible since, from the construction of C', no action of C can be enabled from a closed state.

Observe that shr distributes over combination of transitions. Thus,

$$\mathit{shr}(tr_{q'}^{H'} \restriction \mathit{acts}(M_i \| C)) = \sum_j p_j(\mathit{shr}(q') \frown tr'_j), \tag{7.48}$$

which is generated by a combination of transitions of $M_i \| C$.

We now turn to Θ_o. The set Θ_o can be partitioned into sets $(\Theta_j)_{j \geq 0}$, where each set Θ_j consists of those states q' of Θ_o where a particular state $\bullet_j$ of C' occurs without its matching action a_j. Each element q' of Θ_j can be split into two parts $q_1 \frown q_2$, where $\mathit{lstate}(q_1) \lceil C' = \bullet_j$. Denote q_1 by $\mathit{head}(q')$. Partition Θ_j into other sets $(\Theta_{j,k})_{k \geq 0}$, where each $\Theta_{j,k}$ is an equivalence class of the relation that relates two states iff they have the same head. Denote the common head of the states of $\Theta_{i,j}$ by $\mathit{head}(\Theta_{i,j})$. For each pair of states q_1, q_2 of H' such that $q_1 \leq q_2$, denote by $p_{q_1 q_2}$ the probability value such that $P_{H'}[C_{q_2}^{H'}] = P_{H'}[C_{q_1}^{H'}] p_{q_1 q_2}$. Then, for each equivalence class $\Theta_{i,j}$, the expression

$$\sum_{q' \in \Theta_{j,k}} \bar{p}_{q'}^{\mathit{shr}^{-1}(q)} P_{q'}^{H'}[\mathit{acts}(M_i \| C)] \mathit{shr}(tr_{q'}^{H'} \restriction \mathit{acts}(M_i \| C)) \tag{7.49}$$

can be rewritten into

$$\left(\bar{p}_{\mathit{head}(\Theta_{i,j})}^{\mathit{shr}^{-1}(q)} \sum_{q' \in \Theta_{j,k}} p_{\mathit{head}(q')q'} \right) \sum_{q' \in \Theta_{j,k}} \frac{p_{\mathit{head}(q')q'}}{\sum_{q' \in \Theta_{j,k}} p_{\mathit{head}(q')q'}} P_{q'}^{H'}[a_j] \mathit{shr}(tr_{q'}^{H'} \restriction \mathit{acts}(M_i \| C)) \tag{7.50}$$

where (7.50) is obtained from (7.49) by expressing each $\bar{p}_{q'}^{\mathit{shr}^{-1}(q)}$ as $\bar{p}_{\mathit{head}(q')}^{\mathit{shr}^{-1}(q)} p_{\mathit{head}(q')q'}$, by grouping $\bar{p}_{\mathit{head}(\Theta_{i,j})}^{\mathit{shr}^{-1}(q)}$, which is equal to $\bar{p}_{\mathit{head}(q')}^{\mathit{shr}^{-1}(q)}$ for each q' os $\Theta_{i,j}$, by substituting $P_{q'}^{H'}[a_j]$ for $P_{q'}^{H'}[\mathit{acts}(M_i \| C]$ (action a_j is the only action of $M_i \| C$ that can be performed from q' due to the structure of H'), and by multiplying and dividing by $\sum_{q' \in \Theta_{j,k}} p_{\mathit{head}(q')q'}$.

Observe that each transition that appears in (7.50) is generated by some transitions of $M_i \| C$. Thus, the transition of (7.50) is generated by a combined transition of $M_i \| C$. Denote this transition by $tr_{j,k}$. Then, in Expression (7.47) it is possible to substitute each subexpression $\sum_{q' \in \Theta_{j,k}} \bar{p}_{q'}^{\mathit{shr}^{-1}(q)} P_{q'}^{H'}[\mathit{acts}(M_i \| C)] \mathit{shr}(tr_{q'} \restriction \mathit{acts}(M_i \| C))$ with $(\bar{p}_{\mathit{head}(q')}^{\mathit{shr}^{-1}(q)} \sum_{q' \in \Theta_{j,k}} p_{\mathit{head}(q')q'}) tr_{j,k}$. This is enough to conclude.

b. For each state q of H,

$$P_H[C_q] = \sum_{q' \in min(shr^{-1}(q))} P_{H'}[C_{q'}]. \tag{7.51}$$

This is shown by induction on the length of q. If q consists of a start state only, then the result is trivial. Otherwise, from the definition of the probability of a cone and (7.47),

$$P_H[C_{qas}] = \sum_{q' \in shr^{-1}(q)} P_{H'}[C_{q'}] P_{q'}^{H'}[a \times shr^{-1}(qas)]. \tag{7.52}$$

Observe that the states of $min(shr^{-1}(qas))$ are the states that appear in $(a \times shr^{-1}(qas)) \cap \Omega_{q'}$ for some $q' \in shr^{-1}(q)$. Thus, $P_H[C_{qas}] = \sum_{q' \in min(shr^{-1}(qas))} P_{H'}[C_{q'}]$.

c. $tdistr(H) = tdistr(H') \restriction acts(M_i \| C)$.

Let β be a finite trace of H or the projection of a finite trace of H'. Then $\{\alpha \in \Omega_{H'} \mid \beta \leq trace(\alpha) \restriction acts(M_i \| C)\}$ can be expressed as a union of disjoint cones $\cup_{q \in \Theta} C_q$ where

$$\Theta = \{q \in states(H') \mid trace(q) \restriction acts(M_i \| C) = \beta, lact(q) = lact(\beta)\}. \tag{7.53}$$

Observe that Θ is a set of closed states. The set $shr(\Theta)$ is the set

$$shr(\Theta) = \{q \in states(H) \mid trace(q) = \beta, lact(q) = lact(\beta)\}, \tag{7.54}$$

which is a characterization of $\{\alpha \in \Omega_H \mid \beta \leq trace(\alpha)\}$ as a union of disjoint cones. Observe that $min(shr^{-1}(shr(\Theta))) = \Theta$. Moreover, for each $q_1 \neq q_2$ of $shr(\Theta)$, $shr^{-1}(q_1) \cap shr^{-1}(q_2) = \emptyset$. Thus, from (7.51), $P_{H'}[\cup_{q \in \Theta} C_q] = P_H[q \in shr(\Theta) C_q]$.

To complete the proof of (7.43), it is enough to observe that $H_1 = shr(H_1')$. Property (7.43) is then expressed by property (c).

Suppose by contradiction that it is possible to obtain $\mathcal{D}'$ from $M_2 \| C'$. Consider the scheduler that leads to $\mathcal{D}'$ in $M_2 \| C'$, and let H_2' be the corresponding probabilistic execution. First, we build a new probabilistic execution H_2'' of $M_2 \| C'$ whose trace distribution is $\mathcal{D}'$, such that there is no action of M_2 between each state of the kind $\bullet_i$ and the occurrence of the corresponding external action of C, and such that all the transitions between a state of the kind $\bullet_j$ and the corresponding occurrences of action a_j are scheduled. Then we let $H_2 = shr(H_2'')$. This leads to a contradiction since $tdistr(H_2) = \mathcal{D}$. The rest of the proof is dedicated to the construction of H_2''.

For each state q of H_2', let $shf(q)$ be the set of sequences q' that can be obtained from q as follows: each sequence

$$(s_0, \bullet_j) b_1 (s_1, \bullet) \cdots b_k (s_k, \bullet) a_j (s, c)$$

is replaced with

$$(s_0, \bullet_j) b_{i_1} (s_0, \bullet) \cdots b_{i_l} (s_0, \bullet) a_j (s_0, c) b_{k_1} (s_{k_1}, c) \cdots b_{k_m} (s, c)$$

where $i_1, \ldots, i_l$ is the ordered sequence of the indexes of the b's that are actions of C', and $k_1, \ldots, k_m$ is the ordered sequence of the indexes of the b's that are actions of M_2; each sequence

$$(s_0, \bullet_j) b_1 (s_1, \bullet) \cdots b_k (s_k, \bullet)$$

occurring at the end of q either is replaced with

$$(s_0, \bullet_j) b_{i_1} (s_0, \bullet) \cdots b_{i_l} (s_0, \bullet) \frown \alpha \frown (s_0, \bullet) a_j (s_0, c) b_{k_1} (s_{k_1}, c) \cdots b_{k_m} (s, c)$$

where $i_1, \ldots, i_l$ is the ordered sequence of the indexes of the b's that are actions of C', $k_1, \ldots, k_m$ is the ordered sequence of the indexes of the b's that are actions of M_2, and α, called an *extension* for q, is an arbitrary execution fragment of $M_2 \| C'$ that leads to the occurrence of a_j, or, is replaced with a prefix of $(s_0, \bullet_j) b_{i_1} (s_0, \bullet) \cdots b_{i_l} (s_0, \bullet)$. Then,

$$\mathit{states}(H_2'') \triangleq \bigcup_{q \in \mathit{states}(H_2')} \mathit{shf}(q). \tag{7.55}$$

Let $(q, \mathcal{P})$ be a restricted transition of H_2', and suppose that only actions of M_2 and V_{start} occur. Let q' be a state of $\mathit{shf}(q)$. Then, for each $(a, q_1) \in \Omega$ there is exactly one $q_1' \in \mathit{shf}(q_1)$ such that $q' \leq q_1'$ and $|q_1'| = |q'| + 1$. Denote such q_1' by $\mathit{shf}_{q'}(q_1)$. Let $\Omega' = \{(a, \mathit{shf}_{q'}(q_1) \mid (a, q_1) \in \Omega\}$, and let, for each $(a, q_1') \in \Omega'$, $P'[(a, q_1')] = P[(a \times \mathit{shf}^{-1}(q_1'))]$, where $\mathit{shf}^{-1}(q)$ is the set of states q' of H_2' such that $q \in \mathit{shf}(q')$. Then define the transition $\mathit{shf}_{q'}((q, \mathcal{P}))$ to be

$$\mathit{shf}_{q'}((q, \mathcal{P})) \triangleq (q', \mathcal{P}). \tag{7.56}$$

For each state q of H_2'', let $\mathit{min}(\mathit{shf}^{-1}(q))$ be the set of minimal states of $\mathit{shf}^{-1}(q)$ under prefix ordering. Let q be a closed state of H_2'', and let $q' \in \mathit{shf}^{-1}(q)$. If q' is an open state, then let α be the extension for q' that is used in q, and let $E_{q'}^q$ be the product of the probabilities of the edges of α. For each state q' of $\mathit{shf}^{-1}(q)$, where q is closed, let

- $p_{q'}^q \triangleq P_{H_2'}[C_{q'}]$ if q' is closed;
- $p_{q'}^q \triangleq P_{H_2'}[C_{q'}] E_{q'}^q$ if q' is open.

For each $q' \in \mathit{shf}^{-1}(q)$, let

$$\bar{p}_{q'}^{\mathit{shf}^{-1}(q)} \triangleq \frac{p_{q'}^q}{\sum_{q'' \in \mathit{min}(\mathit{shf}^{-1}(q))} p_{q''}^q}. \tag{7.57}$$

If q is open, then the transition enabled from q in H_2'' is the one due to the transition of C' enabled from $\mathit{lstate}(q) \lceil C'$; if q is closed, then the transition enabled from q in H_2'' is

$$\sum_{q' \in \mathit{shf}^{-1}(q)} \bar{p}_{q'}^{\mathit{shf}^{-1}(q)} P_{q'}^{H_2'}[\mathit{acts}(H_2') \backslash (\mathit{acts}(C) \cup V_2)] \tag{7.58}$$
$$\mathit{shf}_q(\mathit{tr}_{q'}^{H_2'} \upharpoonright (\mathit{acts}(H_2') \backslash (\mathit{acts}(C) \cup V_2))).$$

The probabilistic execution H_2'' satisfies the following properties.

a. H''_2 is a probabilistic execution of $M_2\|C'$.

The fact that each state of H''_2 is reachable can be shown by a simple inductive argument; the fact that each state of H''_2 is a finite execution fragment of $M_2\|C'$ follows from a simple analysis of the definition of *shf*.

We need to check that for each state q of H''_2 the transition enabled from q in H''_2 is generated by a combination of transitions of $M_2\|C'$. If q is an open state, then the result follows immediately from the definition of the transition relation of H''_2. If q is a closed state, then consider a state $q' \in \mathit{shf}^{-1}(q)$. The transition $tr_{q'}^{H'_2} \restriction (acts(H'_2)\backslash V_2)$, which appears in Expression (7.58), can be expressed as $\sum_i p_i(q' \frown tr_i)$, where each tr_i is a transition of $M_2\|C'$ enabled from $lstate(q')$. We distinguish two cases.

1. tr_i is a transition of M_2.
   Then $tr_i = ((s,c), a, \mathcal{P} \otimes \mathcal{D}(c))$ for some action a and probability space $\mathcal{P}$, where $(s,c) = lstate(q')$. Let $lstate(q) = (s', c')$. Then, $s' = s$. Define tr'_i to be the transition $((s,c'), a, \mathcal{P} \otimes \mathcal{D}(c'))$. Then tr'_i is a transition of $M_2\|C'$ and $\mathit{shf}_q(q' \frown tr_i) = q \frown tr'_i$.
2. tr_i is a transition of C'.
   Then $tr_i = ((s,c), a, \mathcal{D}(s) \otimes \mathcal{P})$ for some action a and probability space $\mathcal{P}$, where $(s,c) = lstate(q')$. Let $lstate(q) = (s', c')$. Then, $s' = s$ and $c = c'$ (q is closed). Define tr'_i to be tr_i. Then tr'_i is a transition of $M_2\|C'$ and $\mathit{shf}_q(q' \frown tr_i) = q \frown tr'_i$.

Observe that *shf* distributes over combination of transitions, and thus, the transition $\mathit{shf}_q(t_{q'}^{H'_2} \restriction (acts(H'_2)\backslash V_2))$ can be expressed as $\sum_i p_i(q \frown t'_i)$, which is generated by a combination of transitions of $M_2\|C'$.

b. For each state q of H''_2,

$$P_{H''_2}[C_q] = \begin{cases} \sum_{q' \in min(\mathit{shf}^{-1}(q))} p^q_{q'} & \text{if } q \text{ is closed,} \\ \sum_{q' \in min(\mathit{shf}^{-1}(q))} P_{H'_2}[C_{q'}] & \text{if } q \text{ is open.} \end{cases} \tag{7.59}$$

The proof is by induction on the length of q. If q consists of a start state only, then the result is trivial. Otherwise, consider $P_{H''_2}[C_{qas}]$. We distinguish two cases.

1. q is open.
   In this case a is an action of $V_2 \cup acts(C)$, and each state of $\mathit{shf}^{-1}(q)$ is open. From the definition of the probability of a cone and induction,

$$P_{H''_2}[C_{qas}] = \left(\sum_{q' \in min(\mathit{shf}^{-1}(q))} P_{H'_2}[C_{q'}] \right) P_q^{H''_2}[(a, qas)]. \tag{7.60}$$

   We distinguish two other cases.

   (a) $a \in V_2$.
   Observe that all the states of $min(\mathit{shf}^{-1}(q))$ enable the same transition of C' that is enabled from q. Moreover, for each $q' \in min(\mathit{shf}^{-1}(q))$, action a occurs with probability 1 (in $\mathcal{D}'$ each occurrence of a start action is followed by an

external action with probability 1), and the probability of reaching a state of $min(shf^{-1}(qas))$ given that a occurs is $P_q^{H_2''}[(a, qas)]$ (recall that q enables only action a). Since all the states of $min(shf^{-1}(qas))$ are open and have a prefix in $min(shf^{-1}(q))$, we can conclude

$$P_{H_2''}[C_{qas}] = \sum_{q' \in min(shf^{-1}(qas))} P_{H_2'}[C_{q'}]. \tag{7.61}$$

(b) $a \in acts(C)$.

From the definition of H_2'', $P_q^{H_2''}[(a, qas)] = 1$. Observe that all the states of $min(shf^{-1}(q))$ enable the same transition of C that is enabled from q. Moreover, for each $q' \in min(shf^{-1}(q))$, action a occurs with probability 1 (in $\mathcal{D}'$ each occurrence of a start action is followed by an external action with probability 1), leading to a state of $shf^{-1}(qas)$ for sure (recall that q enables only action a). Thus, for each $q' \in shf^{-1}(q)$,

$$P_{H_2'}[C_{q'}] = \sum_{q'' \in min(shf^{-1}(qas)) | q' \leq q''} P_{H_2'}[C_{q''}]. \tag{7.62}$$

Combining (7.60) and (7.62), we obtain

$$P_{H_2''}[C_{qas}] = \sum_{q' \in min(shf^{-1}(qas))} P_{H_2'}[C_{q'}]. \tag{7.63}$$

For each $q' \in min(shf^{-1}(qas))$, if q' is open, then $p_{q'}^{qas} = P_{H_2'}[C_{q'}]$ by definition; if q' is closed, then $p_{q'}^{qas} = P_{H_2'}[C_{q'}]$ since $E_{q'}^{qas} = 1$ (no α must be added by shf to get q' from qas). Thus, (7.63) becomes

$$P_{H_2''}[C_{qas}] = \sum_{q' \in min(shf^{-1}(qas))} p_{q'}^{qas}. \tag{7.64}$$

2. q is closed.

In this case, from the definition of the probability of a cone and (7.58),

$$P_{H_2''}[C_{qas}] = P_{H_2''}[C_q] \left(\sum_{q' \in shf^{-1}(q)} \bar{p}_{q'}^{shf^{-1}(q)} P_{q'}^{H_2'}[a \times shf^{-1}(qas)] \right) \tag{7.65}$$

From induction, the definition of $\bar{p}_{q'}^{shf^{-1}(q)}$, and an algebraic simplification,

$$\begin{aligned} P_{H_2''}[C_{qas}] \quad = & \sum_{q' \in shf^{-1}(q) | closed(q')} P_{H_2'}[C_{q'}] P_{q'}^{H_2'}[a \times shf^{-1}(qas)] + \\ & \sum_{q' \in shf^{-1}(q) | open(q')} P_{H_2'}[C_{q'}] E_{q'}^{q} P_{q'}^{H_2'}[a \times shf^{-1}(qas)]. \end{aligned} \tag{7.66}$$

We distinguish two subcases.

(a) qas is open.

In this case each state q' of $shf^{-1}(q)$ such that $P_{q'}^{H_2'}[a \times shf^{-1}(qas)] > 0$ is closed, and thus only the first summand of (7.66) is used. Moreover, for each q' of $shf^{-1}(q)$ the set $\Omega_{q'}^{H_2'} \cap a \times shf^{-1}(qas)$ is made of open states $q'as'$ such that $E_{q'as'}^{qas} = 1$. Observe that all the states of $min(shf^{-1}(qas))$ are captured. Thus,

$$P_{H_2''}[C_{qas}] = \sum_{q' \in min(shf^{-1}(qas))} p_{q'}^{q}. \tag{7.67}$$

(b) qas is closed.
In this case, for each $q' \in shf^{-1}(q)$, if q' is closed, then all the states reached in $\Omega_{q'} \cap (\{a\} \times shf^{-1}(qas))$ are closed, and if q' is open, then all the states reached in $\Omega_{q'} \cap (\{a\} \times shf^{-1}(qas))$ are open and the extension α does not change, i.e., the term E does not change. Observe that all the states of $min(shf^{-1}(qas))$ are captured. Thus,

$$P_{H_2''}[C_{qas}] = \sum_{q' \in min(shf^{-1}(qas))} p_{q'}^{q}. \tag{7.68}$$

c. $tdistr(H_2') = tdistr(H_2'')$.

Let β be a finite trace of H_2' or H_2''. Then $\{\alpha \in \Omega_{H_2'} \mid \beta \leq trace(\alpha)\}$ can be expressed as a union of disjoint cones $\cup_{q \in \Theta} C_q$. We distinguish two cases.

1. β does not end with an action of C.
Then

$$\Theta = \{q \in states(H') \mid trace(q) = \beta, lact(q) = lact(\beta)\}. \tag{7.69}$$

The set $\Theta' = \{q \in shf(\Theta) \mid lact(q) = lact(\beta)\}$ is a characterization of $\{\alpha \in \Omega_{H_2''} \mid \beta \leq trace(\alpha)\}$ as a union of disjoint cones. Observe that $min(shf^{-1}(\Theta')) = \Theta$ and that for each $q_1 \neq q_2$ of Θ', $min(shf^{-1}(q_1)) \cap min(shf^{-1}(q_2)) = \emptyset$. Thus, from (7.51), $P_{H_2'}[\{\alpha \in \Omega_{H_2'} \mid \beta \leq trace(\alpha)\}] = P_{H_2''}[\{\alpha \in \Omega_{H_2''} \mid \beta \leq trace(\alpha)\}]$.

2. β ends with an action of C.
In this case $\beta = \beta' a_j$ for some action $a_j \in acts(C)$. Since in H_2' and H_2'' after the occurrence of a state $\bullet_j$ the corresponding action a_j occurs with probability 1, we can assume that all the states of Θ end in $\bullet_j$, i.e.,

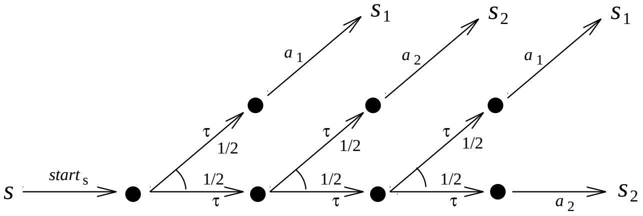
$$\Theta = \{q \in states(H') \mid trace(q) = \beta', \text{ and } lstate(q) \text{ is one of the } \bullet_j\text{'s}\}. \tag{7.70}$$

Then the set $\Theta' = min(shf(\Theta))$ is a characterization of $\{\alpha \in \Omega_{H_2''} \mid \beta \leq trace(\alpha)\}$ as a union of disjoint cones. Observe that all the elements of Θ are open. Property (7.59) is sufficient to conclude. ■

Lemma 7.5.7 *Let C be a distinguishing binary separated context for two probabilistic automata M_1 and M_2. Then there exists a distinguishing total binary separated context C' for M_1 and M_2 where all the probabilistic transitions have a uniform distribution. C' is called a* balanced separated context.

Proof. We achieve the result in two steps. First we decompose a binary probabilistic transition into several binary uniform probabilistic transitions, leading to a new distinguishing context C_1; then we use Lemma 7.5.4 to make C_1 into a cycle-free context.

The context C_1 is obtained from C by expressing each probabilistic transition of C by means of, possibly infinitely many, binary probabilistic transitions. For each state s of C, let $start_s$ be a new action. If s enables a probabilistic transition with actions a_1, a_2 to states s_1, s_2, respectively, and with probabilities p_1, p_2, respectively, then C_1 enables from s a deterministic transition with action $start_s$. Then, C_1 enables an internal probabilistic transition with a uniform distribution. If $p_1 > p_2$ ($p_2 > p_1$), then one of the states that is reached enables a deterministic transition with action a_1 (a_2). The other state enables a new internal probabilistic transition with a uniform binary distribution, and the transitions from the successive states are determined by giving a_1 probability $2(p_1 - 1/2)$ and a_2 probability $2p_2$ (a_1 probability $2p_1$ and a_2 probability $2(p_2 - 1/2)$). If $p_1 = p_2$, then one state enables a_1, and the other state enables a_2. For example, if $p_1 = 5/8$ and $p_2 = 3/8$, then the corresponding transitions of C_1 are represented below. Let $\mathcal{D}$ be a trace distribution of $M_1 \| C$ that is not a trace distribution of $M_2 \| C$. Consider a probabilistic execution H_1 of $M_1 \| C$ whose trace distribution is $\mathcal{D}$, and consider the scheduler that leads to H_1 in $M_1 \| C$. Apply to $M_1 \| C_1$ the same scheduler with the following modification: whenever a probabilistic transition of C is scheduled, schedule the *start* action from C_1, then schedule the internal transitions to resolve the probabilistic choice, and finally schedule the chosen action. Denote the resulting probabilistic execution by H_1' and the resulting trace distribution by $\mathcal{D}'$. Then,



$$\mathcal{D}' \upharpoonright acts(M_1 \| C) = \mathcal{D}. \tag{7.71}$$

To prove (7.71), we define a new construction shr_1, similar to *shr*, to be applied to probabilistic executions of $M_i \| C_1$ such that no action of M_i occurs between the occurrence of a $start_s$ action and the occurrence of one of the corresponding external actions of C, and such that all the transitions of C_1 between the occurrence of an action $start_s$ and the occurrence of one of the corresponding external actions of C are scheduled. The new function is identical to *shr* if we consider each state reached immediately after the occurrence of a start action like the states $\bullet_j$ used in Lemma 7.5.6. We leave the details to the reader.

Suppose by contradiction that it is possible to obtain $\mathcal{D}'$ from $M_2 \| C_1$. Consider the scheduler that leads to $\mathcal{D}'$ in $M_2 \| C_1$, and let H_2' be the corresponding probabilistic execution. First, we build a new probabilistic execution H_2'' of $M_2 \| C_1$ whose trace distribution is $\mathcal{D}'$, such that no action of M_i occurs between the occurrence of a $start_s$ action and the occurrence of one of the corresponding external action of C, and such that all the transitions of C_1 between the occurrence of an action $start_s$ and the occurrence of one of the corresponding external action of C are scheduled. Then we let $H_2 = shr_1(H_2'')$. This leads to a contradiction since $tdistr(H_2) = \mathcal{D}$.

The construction of H_2'', which is left to the reader, is the same as *shf* if we consider each state reached immediately after the occurrence of a start action like the states $\bullet_j$ used in Lemma 7.5.6. ■

Lemma 7.5.8 *Let C be a distinguishing balanced separated context for two probabilistic automata M_1 and M_2. Then there exists a distinguishing binary separated context C' for M_1 and M_2 with no internal actions and such that each action appears exactly in one edge of the transition tree. C' is called a* total balanced separated context.

Proof. The context C' is obtained from C by renaming all of its actions so that each edge of the new transition relation has its own action.

Let $\mathcal{D}$ be a trace distribution of $M_1\|C$ that is not a trace distribution of $M_2\|C$. Consider a probabilistic execution H_1 of $M_1\|C$ whose trace distribution is $\mathcal{D}$, and consider the scheduler that leads to H_1 in $M_1\|C$. Apply to $M_1\|C'$ the same scheduler with the following modification: whenever a transition of C is scheduled, schedule the corresponding transition of C'. Denote the resulting probabilistic execution by H_1' and the corresponding trace distribution by $\mathcal{D}'$. From construction, H_1 and H_1' are the same up to the names of the actions of C. Thus, if ρ' is the function that maps each action of C' to its original name in C, $\mathcal{D} = \rho'(\mathcal{D}')$ (the renaming of a trace distribution is the probability space induced by the function that renames traces).

Suppose by contradiction that it is possible to obtain $\mathcal{D}'$ from $M_2\|C'$. Consider the scheduler that leads to $\mathcal{D}'$ in $M_2\|C'$, and let H_2' be the corresponding probabilistic execution. Apply to $M_2\|C$ the same scheduler with the following modifications: whenever a transition of C' is scheduled, schedule the corresponding transition of C with the unrenamed actions. Let H_2 be the resulting probabilistic execution. From the construction, H_2 and H_2' are the same up to the names of the actions of C. Thus, $\mathit{tdistr}(H_2) = \rho'(\mathcal{D}') = \mathcal{D}$, which is a contradiction. ■

Lemma 7.5.9 *Let C be a distinguishing total balanced separated context for two probabilistic automata M_1 and M_2. Then there exists a distinguishing total balanced separated context C' for M_1 and M_2 that from every state enables two deterministic transitions and a probabilistic transition with a uniform distribution over two choices. C' is called a* complete context.

Proof. In this case it is enough to complete C by adding all the missing transitions and states. If $\mathcal{D}$ is a trace distribution of $M_1\|C$ that is not a trace distribution of $M_2\|C$, then it is enough to use on $M_1\|C'$ the same scheduler that is used in $M_1\|C$. In fact, since each new transition of C' has a distinct action, none of the new transitions of C' can be used in $M_2\|C'$ to generate $\mathcal{D}$. ■

Lemma 7.5.10 *Let C be a distinguishing complete context for two probabilistic automata M_1 and M_2. Then the principal context C_P is a distinguishing context for M_1 and M_2.*

Proof. The result is achieved in two steps. First the actions of C are renamed so that each state enables two deterministic transitions with actions *left* and *right*, respectively, and a probabilistic transition with actions *pleft* and *pright*. Call this context C_1. Then, by observing that each state s of C_1 is uniquely determined by the trace of the unique execution of C_1 that leads to s, all the states of C_1 are collapsed into a unique one.

Thus, we need to show only that C_1 is a distinguishing context. Let $\mathcal{D}$ be a trace distribution of $M_1\|C$ that is not a trace distribution of $M_2\|C$. Consider the scheduler that leads to $\mathcal{D}$ in $M_1\|C$, and apply to $M_1\|C_1$ the same scheduler with the following modification: whenever a transition of C is scheduled, schedule the corresponding transition of C_1. Denote the resulting trace distribution by $\mathcal{D}'$. Note that if we rename all the actions of C_1 into their original name in C, then we obtain $\mathcal{D}$.

Suppose by contradiction that it is possible to obtain $\mathcal{D}'$ from $M_2\|C_1$. Consider the scheduler that leads to $\mathcal{D}'$ in $M_2\|C_1$, and apply to $M_2\|C$ the same scheduler with the following modification: whenever a transition of C_1 is scheduled, schedule the corresponding transition of C. The resulting trace distribution is $\mathcal{D}$, which is a contradiction. ■

Lemma 7.5.11 *Let C_P be a distinguishing context for two probabilistic automata M_1 and M_2. Then the simple principal context, denoted by C, is a distinguishing context for M_1 and M_2.*

Proof. Let $\mathcal{D}$ be a trace distribution of $M_1 \| C_P$ that is not a trace distribution of $M_2 \| C_P$. Consider a probabilistic execution H_1 of $M_1 \| C_P$ whose trace distribution is $\mathcal{D}$, and consider the scheduler that leads to H_1 in $M_1 \| C_P$. Apply to $M_1 \| C$ the same scheduler with the following modification: whenever the probabilistic transition of C_P is scheduled, schedule the *start* action of C followed by the next transition of C that becomes enabled. Denote the resulting probabilistic execution by H_1' and the resulting trace distribution by $\mathcal{D}'$. Then,

$$\mathcal{D}' \upharpoonright acts(M_1 \| C_P) = \mathcal{D}. \tag{7.72}$$

To prove (7.72), we define a new construction shr_2, similar to shr, to be applied to probabilistic executions of $M_i \| C$ such that no action of M_i occurs between the occurrence of a *start* action and the occurrence of one of the actions *pleft* and *pright*, and such that the transitions labeled with *pleft* and *pright* occur whenever they are enabled. The new function is identical to *shr* if we consider each state reached after an action *start* as a state of the kind $\bullet_j$. We leave the details to the reader.

Suppose by contradiction that it is possible to obtain $\mathcal{D}'$ from $M_2 \| C$. Consider the scheduler that leads to $\mathcal{D}'$ in $M_2 \| C$, and let H_2' be the corresponding probabilistic execution. First, we build a new probabilistic execution H_2'' of $M_2 \| C$ whose trace distribution is $\mathcal{D}'$, such that no action of M_2 occurs between the occurrence of a *start* action and the occurrence of one of the actions *pleft* and *pright*, and such that the transitions labeled with *pleft* and *pright* occur whenever they are enabled. Then we let $H_2 = clp_2(H_2'')$. This leads to a contradiction since $tdistr(H_2) = \mathcal{D}$.

The construction of H_2'', which is left to the reader, is the same as *shf* if we consider each state reached immediately after the occurrence of a start action like the states $\bullet_j$ used in Lemma 7.5.6. ■

Proof of Theorem 7.5.1. Let $M_1 \sqsubseteq_{DC} M_2$. Then, from Lemma 7.5.11, $M_1 \| C_P \sqsubseteq_D M_2 \| C_P$. Conversely, let $M_1 \| C_P \sqsubseteq_D M_2 \| C_P$. Then, from Lemmas 7.5.3, 7.5.4, 7.5.5, 7.5.6, 7.5.7, 7.5.8, 7.5.9, and 7.5.10, $M_1 \sqsubseteq_{DC} M_2$. ■

7.6 Discussion

A trace-based semantics similar to ours is studied for generative processes by Jou and Smolka [JS90]. One of the processes of Jou and Smolka is essentially one of our probabilistic executions. The semantics of a process is given by a function, called a trace function, that associates a probability with each finite trace. Since our trace distributions are determined by the probabilities of the cones, our trace distributions are characterized completely by the trace functions of Jou and Smolka. In other words, the trace semantics of Jou and Smolka is the semantics that we use to say that two probabilistic executions have the same trace distribution.

Jou and Smolka define also a notion of a maximal trace function. Given a probabilistic execution H, the interpretation of a maximal trace function in our framework is a function that associates with each finite trace β the probability of the extended executions on Ω_H that end in δ and whose trace is β. Jou and Smolka show that the trace function of a process is sufficient to determine the maximal trace function of the process. In our trace distributions the maximal trace function of a probabilistic execution is given by the probability of each finite trace in the corresponding trace distribution. From the definition of a trace distribution the probability of each finite trace is determined uniquely by the probabilities of the cones, and thus the result of Jou and Smolka holds also in our framework.

Chapter 8

Hierarchical Verification: Simulations

8.1 Introduction

In Chapter 7 we have studied the trace distribution precongruence as an instance of the hierarchical method for the verification of probabilistic systems. Another instance of the hierarchical method is called the *simulation method*. According to the simulation method, rather than comparing two probabilistic automata through some abstract observations, two probabilistic automata are compared by establishing some relation between their states and by showing that the two probabilistic automata can simulate each other via the given relation. Standard work on simulation relations appears in [Mil89, Jon91, LV91]. Simulation relations are stronger than the trace preorder, and are often used as a sound proof technique for the trace preorder.

In this chapter we study how to extend some of the relations of [Mil89, Jon91, LV91] to the probabilistic framework. We start with the generalization of the simplest relations that do not abstract from internal computation, and we conclude with the generalization of the forward simulations of [LV91] that approximate closely the trace distribution preorder. We prove the equivalent of the Execution Correspondence Lemma [GSSL94] for probabilistic automata, which states that there is a strong connection between the probabilistic executions of two probabilistic automata related by some simulation relation. Finally, we use the new Execution Correspondence Lemma to prove that the existence of a probabilistic forward simulation is sufficient to prove the trace distribution precongruence relation.

8.2 Strong Simulations

One of the finest equivalence relations for ordinary automata would be graph isomorphism; however, it is widely recognized that graph isomorphism distinguishes too much. A coarser equivalence relation is strong bisimulation [Par81, Mil89], where two automata A_1 and A_2 are equivalent iff there is an equivalence relation between their states so that for each pair (s_1, s_2) of equivalent states,

if $s_1 \xrightarrow{a} s_1'$, then there exists a state s_2' equivalent to s_1' such that $s_2 \xrightarrow{a} s_2'$.

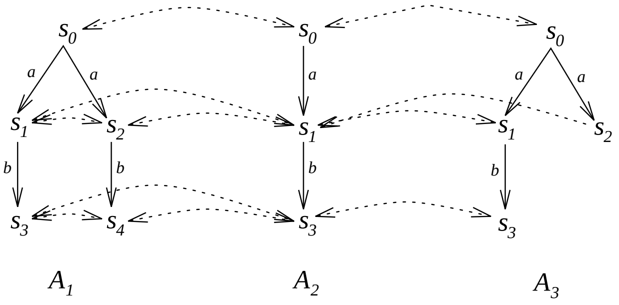


Figure 8-1: The difference between strong bisimulation and the kernel of strong simulation.

That is, A_1 and A_2 simulate each other. A preorder relation that is closely connected to strong bisimulation is strong simulation. An automaton A_1 is strongly simulated by another automaton A_2 iff there is a relation between the states of A_1 and the states of A_2 so that for each pair (s_1, s_2) of related states,

if $s_1 \xrightarrow{a} s_1'$, then there exists a state s_2' such that $s_2 \xrightarrow{a} s_2'$ and s_1' is related to s_2'.

The kernel of strong simulation is an equivalence relation that is coarser than bisimulation.

Example 8.2.1 (Strong simulation and strong bisimulation) Figure 8-1 shows the difference between strong bisimulation and the kernel of strong simulation. The double-arrow dotted links represent a strong bisimulation between A_1 and A_2; thus, A_1 and A_2 are strongly bisimilar. There is also a strong simulation from A_2 to A_3, expressed by the dotted lines that have an arrow pointing to A_3, and a strong simulation from A_3 to A_2, expressed by the dotted lines that have an arrow pointing to A_2. Thus, A_2 and A_3 are equivalent according to the kernel of strong simulation. However, there is no bisimulation between A_2 and A_3 since state s_2 of A_3 must be related to state s_1 of A_2 in order for A_2 to be able to simulate the transition $s_0 \xrightarrow{a} s_2$ of A_3, but then it is not possible to simulate the transition $s_1 \xrightarrow{b} s_3$ of A_2 from s_2 in A_3. ■

The extension of strong bisimulation and strong simulation to the probabilistic framework presents a problem due to the fact that a probabilistic transition leads to a probability distribution over states rather than to a single state. Thus, a relation over states needs to be lifted to distributions over states. Here we borrow an idea from [JL91].

Let $\mathcal{R} \subseteq X \times Y$ be a relation between two sets X, Y, and let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two probability spaces of *Probs*(X) and *Probs*(Y), respectively. Then $\mathcal{P}_1$ and $\mathcal{P}_2$ are in relation $\sqsubseteq_{\mathcal{R}}$, written $\mathcal{P}_1 \sqsubseteq_{\mathcal{R}} \mathcal{P}_2$, iff there exists a *weight function* $w : X \times Y \rightarrow [0,1]$ such that

1. for each $x \in X$, $\sum_{y \in Y} w(x,y) = P_1[x]$,
2. for each $y \in Y$, $\sum_{x \in X} w(x,y) = P_2[y]$,
3. for each $(x,y) \in X \times Y$, if $w(x,y) > 0$ then $x \;\mathcal{R}\; y$.

Example 8.2.2 (Lifting of one relation) The idea behind the definition of $\sqsubseteq_{\mathcal{R}}$ is that each state of Ω_1 must be represented by some states of Ω_2, and similarly, each state of Ω_2 must represent one or more states of Ω_1. Figure 8-2 gives an example of two probability spaces that

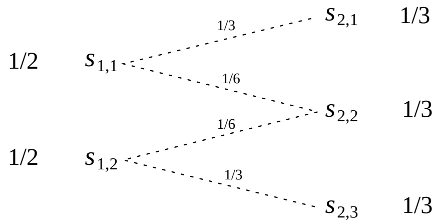


Figure 8-2: Lifting one relation.

are related. The dotted lines connect states that are related by $\mathcal{R}$. Thus, state $s_{1,1}$ can be represented by $s_{2,1}$ for a third of its probability, and by $s_{2,2}$ for the reminder. Similarly, state $s_{2,2}$ represents $s_{1,1}$ for one sixth of its probability and $s_{1,2}$ for the reminder. A useful property of $\sqsubseteq_{\mathcal{R}}$ is its preservation over combination of probability spaces. ■

If $\mathcal{R}$ is an equivalence relation, then we denote the relation $\sqsubseteq_{\mathcal{R}}$ alternatively by $\equiv_{\mathcal{R}}$. The reason for the alternative notation is that whenever $\mathcal{R}$ is an equivalence relation and $\mathcal{P}_1 \equiv_{\mathcal{R}} \mathcal{P}_2$, each equivalence class of $\mathcal{R}$ is assigned the same probability in $\mathcal{P}_1$ and $\mathcal{P}_2$ (cf. Lemma 8.2.2).

The definition of strong bisimulation and strong simulation for probabilistic automata are now straightforward. For convenience assume that M_1 and M_2 do not have common states. A *strong bisimulation* between two simple probabilistic automata M_1, M_2 is an equivalence relation $\mathcal{R}$ over $states(M_1) \cup states(M_2)$ such that

1. each start state of M_1 is related to at least one start state of M_2, and vice versa;

2. for each pair of states $s_1 \mathcal{R} s_2$ and each transition $s_1 \xrightarrow{a} \mathcal{P}_1$ of either M_1 or M_2, there exists a transition $s_2 \xrightarrow{a} \mathcal{P}_2$ of either M_1 or M_2 such that $\mathcal{P}_1 \equiv_{\mathcal{R}} \mathcal{P}_2$.

We write $M_1 \simeq M_2$ whenever $acts(M_1) = acts(M_2)$ and there is a strong bisimulation between M_1 and M_2.

A *strong simulation* between two simple probabilistic automata M_1, M_2 is a relation $\mathcal{R} \subseteq states(M_1) \times states(M_2)$ such that

1. each start state of M_1 is related to at least one start state of M_2;

2. for each pair of states $s_1 \mathcal{R} s_2$ and each transition $s_1 \xrightarrow{a} \mathcal{P}_1$ of M_1, there exists a transition $s_2 \xrightarrow{a} \mathcal{P}_2$ of M_2 such that $\mathcal{P}_1 \sqsubseteq_{\mathcal{R}} \mathcal{P}_2$.

We write $M_1 \sqsubseteq_{\mathrm{SS}} M_2$ whenever $acts(M_1) = acts(M_2)$ and there is a strong simulation from M_1 to M_2. We denote he kernel of strong simulation by $\equiv_{\mathrm{SS}}$. Because of Lemma 8.2.2, our strong bisimulations are the same as the bisimulations of [Han94], and our strong simulations are a generalization of the simulations of [JL91].

It is easy to check that $\simeq$ is an equivalence relation, that $\sqsubseteq_{\mathrm{SS}}$ is a preorder relation, and that both $\simeq$ and $\sqsubseteq_{\mathrm{SS}}$ are preserved by the parallel composition operator.

We conclude this section by proving two results about the lifting of a relation. The first result shows that the lifting of a relation is preserved by the combination of probability spaces; the second result shows that $\mathcal{P}_1 \equiv_{\mathcal{R}} \mathcal{P}_2$ iff $\mathcal{P}_1$ and $\mathcal{P}_2$ assign the same probability to each equivalence class of $\mathcal{R}$.

Lemma 8.2.1 *Let $\mathcal{P}_{X,i} \sqsubseteq_{\mathcal{R}} \mathcal{P}_{Y,i}$ via a weight function w_i, and let $\{p_i\}_{i\geq 0}$ be a family of real numbers between 0 and 1 such that $\sum_{i\geq 0} p_i = 1$. Then $\sum_{i\geq 0} p_i \mathcal{P}_{X,i} \sqsubseteq_{\mathcal{R}} \sum_{i\geq 0} p_i \mathcal{P}_{Y,i}$ via $\sum_{i\geq 0} p_i w_i$.*

Proof. Let $\mathcal{P}_X = \sum_{i\geq 0} p_i \mathcal{P}_{X,i}$, $\mathcal{P}_Y = \sum_{i\geq 0} p_i \mathcal{P}_{Y,i}$, and $w = \sum_{i\geq 0} p_i w_i$. Let $x \in \Omega_X$. Then $\sum_{y\in\Omega_Y} w(x,y) = \sum_{y\in\Omega_Y} \sum_{i\geq 0} p_i w_i(x,y) = \sum_{i\geq 0} p_i (\sum_{y\in\Omega_Y} w_i(x,y)) = \sum_{i\geq 0} p_i P_{X,i}[x] = P_X[x]$. Condition 2 of the definition of $\sqsubseteq_{\mathcal{R}}$ is verified similarly. For Condition 3, let $w(x,y) > 0$. Then there exists an i such that $w_i(x,y) > 0$, and thus $x \; \mathcal{R} \; y$. ■

Lemma 8.2.2 *Let X, Y be two disjoint sets, $\mathcal{R}$ be an equivalence relation on $X \cup Y$, and let $\mathcal{P}_1$ and $\mathcal{P}_2$ be probability spaces of Probs(X) and Probs(Y), respectively. Then, $\mathcal{P}_1 \equiv_{\mathcal{R}} \mathcal{P}_2$ iff for each equivalence class C of $(X\cup Y)/\mathcal{R}$, $P_1[C\cap\Omega_1] = P_2[C\cap\Omega_2]$.*

Proof. Suppose that $\mathcal{P}_1 \equiv_{\mathcal{R}} \mathcal{P}_2$, and let w be the corresponding weight function. Then, for each equivalence class C of $(X\cup Y)/\mathcal{R}$,

$$P_1[C\cap\Omega_1] = \sum_{x\in C\cap\Omega_1} P_1[x] = \sum_{x\in C\cap\Omega_1} \sum_{y\in C\cap\Omega_2} w(x,y), \tag{8.1}$$

and

$$P_2[C\cap\Omega_2] = \sum_{y\in C\cap\Omega_2} P_2[y] = \sum_{y\in C\cap\Omega_2} \sum_{x\in C\cap\Omega_1} w(x,y). \tag{8.2}$$

From the commutativity and associativity of sum,

$$P_1[C\cap\Omega_1] = P_2[C\cap\Omega_2]. \tag{8.3}$$

Conversely, suppose that each equivalence class $(X\cup Y)/\mathcal{R}$ has the same probability in $\mathcal{P}_1$ and $\mathcal{P}_2$. We define $w(x,y)$ for each equivalence class of $(X\cup Y)/\mathcal{R}$, and we assume implicitly that w is 0 for all the pairs $(x,y) \in \Omega_1 \times \Omega_2$ that are not considered in the construction below. Consider any equivalence class C of $(X\cup Y)/\mathcal{R}$, and let $X' = C\cap\Omega_1$, and $Y' = C\cap\Omega_2$. From hypothesis we know that $P_1[X'] = P_2[Y']$. Let $x_1, x_2, \ldots$ be an enumeration of the points of X', and let $y_1, y_2, \ldots$ be an enumeration of the points of Y'. For each i, let $p_i = \sum_{k<i} P_1[x_i]$ and let $q_i = \sum_{k<i} P_2[y_i]$. Then

$$w(x_i, y_j) = \begin{cases} 0 & \text{if } p_{i+1} \leq q_j \text{ or } q_{j+1} \leq p_i \\ min(p_{i+1}, q_{j+1}) - max(p_i, q_j) & \text{otherwise.} \end{cases}$$

Informally, the construction above works as follows. Consider two intervals $[0, P_1[X']]$, and mark the first interval with the points p_i and the second interval with the points q_j. Each interval $[p_i, p_{i+1}]$ has length $P_1[x_i]$ and each interval $[q_j, q_{j+1}]$ has length $P_2[y_j]$. The weight function $w(x_i, y_j)$ is defined to be the length of the intersection of the intervals associated with x_i and y_j, respectively. It is simple to verify that w is a weight function for $\mathcal{P}_1$ and $\mathcal{P}_2$. ■

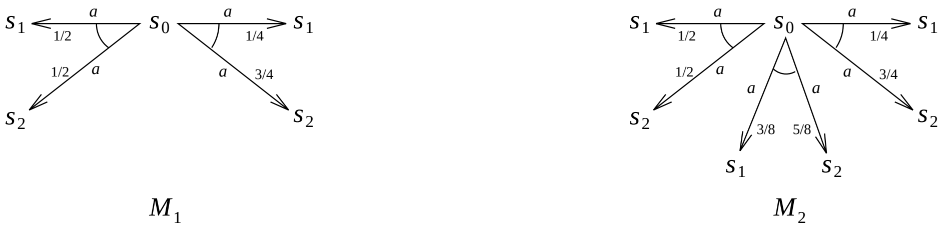


Figure 8-3: Combining transitions to simulate a transition.

8.3 Strong Probabilistic Simulations

In the definition of strong bisimulations and strong simulations we have not taken into account the fact that the nondeterminism can be resolved by combining several transitions probabilistically into a unique one. That is, a transition of a probabilistic automaton could be simulated by combining several transitions of another probabilistic automaton.

Example 8.3.1 (Combining transitions to simulate another transition) Consider the two probabilistic automata M_1 and M_2 of Figure 8-3. M_2 contains the transitions of M_1 plus a transitions that is obtained by combining probabilistically the transitions of M_1. For this reason there is no simulation from M_2 to M_1 (the additional transition cannot be simulated). On the other hand, M_1 and M_2 have exactly the same probabilistic executions, and therefore we do not see any reason to distinguish them. ■

Example 8.3.1 suggests two new relations, which are coarser than strong bisimulation and strong simulation, where the only difference is that a transition can be simulated by a probabilistic combination of transitions.

For convenience assume that M_1 and M_2 do not have common states. A *strong probabilistic bisimulation* between two simple probabilistic automata M_1, M_2 is an equivalence relation $\mathcal{R}$ over $states(M_1) \cup states(M_2)$ such that

1. each start state of M_1 is related to at least one start state of M_2, and vice versa;

2. for each pair of states $s_1 \ \mathcal{R}\ s_2$ and each transition $s_1 \xrightarrow{a} \mathcal{P}_1$ of either M_1 or M_2, there exists a combined transition $s_2 \xrightarrow{a}_{\rm C} \mathcal{P}_2$ of either M_1 or M_2 such that $\mathcal{P}_1 \equiv_{\mathcal{R}} \mathcal{P}_2$.

We write $M_1 \simeq_{\rm P} M_2$ whenever $acts(M_1) = acts(M_2)$ and there is a strong probabilistic bisimulation between M_1 and M_2.

A *strong probabilistic simulation* between two simple probabilistic automata M_1 and M_2 is a relation $\mathcal{R} \subseteq states(M_1) \times states(M_2)$ such that

1. each start state of M_1 is related to at least one start state of M_2;

2. for each pair of states $s_1 \ \mathcal{R}\ s_2$ and each transition $s_1 \xrightarrow{a} \mathcal{P}_1$ of M_1, there exists a combined transition $s_2 \xrightarrow{a}_{\rm C} \mathcal{P}_2$ of M_2 such that $\mathcal{P}_1 \sqsubseteq_{\mathcal{R}} \mathcal{P}_2$.

We write $M_1 \sqsubseteq_{\rm SPS} M_2$ whenever $acts(M_1) = acts(M_2)$ and there is a strong probabilistic simulation from M_1 to M_2. We denote the kernel of strong probabilistic simulation by $\equiv_{\rm SPS}$.

It is easy to check that $\simeq_{\mathrm{P}}$ is an equivalence relation, that $\sqsubseteq_{\mathrm{SPS}}$ is a preorder relation, and that both $\simeq_{\mathrm{P}}$ and $\sqsubseteq_{\mathrm{SPS}}$ are preserved by the parallel composition operator. It is easy as well to verify that a strong bisimulation is also a strong probabilistic bisimulation and that a strong simulation is also a strong probabilistic simulation.

8.4 Weak Probabilistic Simulations

The abstraction from internal computation can be obtained in the same way as for ordinary automata: a transition of a probabilistic automaton should be simulated by a collection of internal and external transitions of another probabilistic automaton. For the formal definition we use the weak combined transitions of Chapter 4.

For convenience assume that M_1 and M_2 do not have common states. A *weak probabilistic bisimulation* between two simple probabilistic automata M_1 and M_2 is an equivalence relation $\mathcal{R}$ over $states(M_1) \cup states(M_2)$ such that

1. each start state of M_1 is related to at least one start state of M_2, and vice versa;
2. for each pair of states $s_1 \ \mathcal{R} \ s_2$ and each transition $s_1 \xrightarrow{a} \mathcal{P}_1$ of either M_1 or M_2, there exists a weak combined transition $s_2 \overset{a\upharpoonright ext(M_2)}{\Longrightarrow_{\mathrm{C}}} \mathcal{P}_2$ of either M_1 or M_2 such that $\mathcal{P}_1 \equiv_{\mathcal{R}} \mathcal{P}_2$.

We write $M_1 =_{\mathrm{P}} M_2$ whenever $ext(M_1) = ext(M_2)$ and there is a weak probabilistic bisimulation between M_1 and M_2.

A *weak probabilistic simulation* between two simple probabilistic automata M_1 and M_2 is a relation $\mathcal{R} \subseteq states(M_1) \times states(M_2)$ such that

1. each start state of M_1 is related to at least one start state of M_2;
2. for each pair of states $s_1 \ \mathcal{R} \ s_2$ and each transition $s_1 \xrightarrow{a} \mathcal{P}_1$ of M_1, there exists a weak combined transition $s_2 \overset{a\upharpoonright ext(M_2)}{\Longrightarrow_{\mathrm{C}}} \mathcal{P}_2$ of M_2 such that $\mathcal{P}_1 \sqsubseteq_{\mathcal{R}} \mathcal{P}_2$.

We write $M_1 \sqsubseteq_{\mathrm{WPS}} M_2$ whenever $ext(M_1) = ext(M_2)$ and there is a weak probabilistic simulation from M_1 to M_2. We denote the kernel of weak probabilistic simulation by $\equiv_{\mathrm{WPS}}$.

It is easy to verify that a strong probabilistic bisimulation is also a weak probabilistic bisimulation and that a strong probabilistic simulation is also a weak probabilistic simulation. However, it is not as easy to verify that $=_{\mathrm{P}}$ is an equivalence relation, that $\sqsubseteq_{\mathrm{WPS}}$ is a preorder relation, and that both $=_{\mathrm{P}}$ and $\sqsubseteq_{\mathrm{WPS}}$ are preserved by the parallel composition operator. The verification of these properties is a simplification of the verification of the same properties for the relation of the next section. For this reason we omit the proofs from this section.

8.5 Probabilistic Forward Simulations

One of the main results of this chapter is that all the relations presented so far are sound for the trace distribution precongruence. However, none of the relations of the previous sections allow for one probabilistic operation to be implemented by several probabilistic operations.

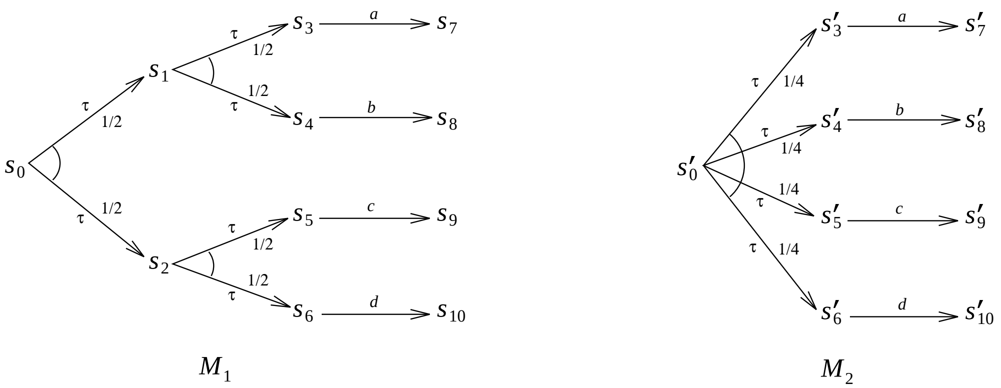


Figure 8-4: Implementation of a probabilistic transition with several probabilistic transitions.

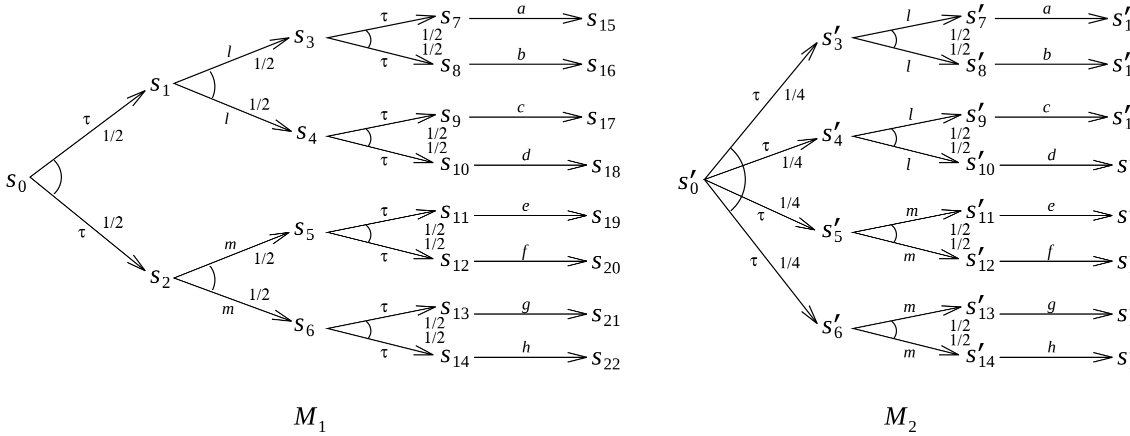


Figure 8-5: A more sophisticated implementation.

Example 8.5.1 (Weak probabilistic simulations are too coarse) Consider the two probabilistic automata of Figure 8-4. The probabilistic automaton M_2, which chooses internally one element out of four with probability $1/4$ each, is implemented by the probabilistic automaton M_1, which flips two fair coins to make the same choice. However, the first transition of M_1 cannot be simulated by M_2 since the probabilistic choice of M_2 is not resolved completely yet in M_1. This situation suggests a new preorder relation where a state of M_1 can be related to a probability distribution over states of M_2. The informal idea behind a relation $s_1 \mathcal{R} \mathcal{P}_2$ is that s_1 represents an intermediate stage of M_1 in reaching the distribution $\mathcal{P}_2$. For example, in Figure 8-4 state s_1 would be related to a uniform distribution $\mathcal{P}$ over states s'_3 and s'_4 ($\mathcal{P} = \mathcal{U}(s'_3, s'_4)$), meaning that s_1 is an intermediate stage of M_1 in reaching the distribution $\mathcal{P}$.

It is also possible to create examples where the relationship between s and $\mathcal{P}$ does not mean simply that s is an intermediate stage of M_1 in reaching the distribution $\mathcal{P}$, but rather that s is an intermediate stage in reaching a probability distribution that can be reached from $\mathcal{P}$. Consider the two probabilistic automata of Figure 8-5. Although not evident at the moment, M_1 and M_2 are in the trace distribution precongruence relation, i.e., $M_1 \sqsubseteq_{DC} M_2$. Following the same idea as for the example of Figure 8-4, state s_1 is related to $\mathcal{U}(s'_3, s'_4)$. However, s_1 is not an intermediate stage of M_1 in reaching $\mathcal{U}(s'_3, s'_4)$, since s_1 enables a transition labeled with an external action l, while in M_2 no external action occurs before reaching $\mathcal{U}(s'_3, s'_4)$. Rather, from s'_3 and s'_4 there are two transitions labeled with l, and thus the only way to simulate the transition $s_1 \xrightarrow{l} \mathcal{U}(s_3, s_4)$ from $\mathcal{U}(s'_3, s'_4)$ is to perform the two transitions labeled with l, which lead to the distribution $\mathcal{U}(s'_7, s'_8, s'_9, s'_{10})$. Now the question is the following: in what sense does $\mathcal{U}(s'_7, s'_8, s'_9, s'_{10})$ represent $\mathcal{U}(s_3, s_4)$? The first observation is that s_3 can be seen as an intermediate stage in reaching $\mathcal{U}(s'_7, s'_8)$, and that s_4 can be seen as an intermediate stage in reaching $\mathcal{U}(s'_9, s'_{10})$. Thus, s_3 is related to $\mathcal{U}(s'_7, s'_8)$ and s_4 is related to $\mathcal{U}(s'_9, s'_{10})$. The second observation is that $\mathcal{U}(s'_7, s'_8, s'_9, s'_{10})$ can be expressed as $1/2\mathcal{U}(s'_7, s'_8) + 1/2\mathcal{U}(s'_9, s'_{10})$. Thus, $\mathcal{U}(s'_7, s'_8, s'_9, s'_{10})$ can be seen as a combination of two probability spaces, each one representing an element of $\mathcal{U}(s_3, s_4)$. This recalls the lifting of a relation that we introduced at the beginning of this chapter. ■

Based on Example 8.5.1, we can move to the formal definition of a probabilistic forward simulation. A *probabilistic forward simulation* between two simple probabilistic automata M_1 and M_2 is a relation $\mathcal{R} \subseteq \mathit{states}(M_1) \times \mathit{Probs}(\mathit{states}(M_2))$ such that

1. each start state of M_1 is related to at least one Dirac distribution over a start state of M_2;
2. for each $s \mathrel{\mathcal{R}} \mathcal{P}'$, if $s \xrightarrow{a} \mathcal{P}_1$, then
   (a) for each $s' \in \Omega'$ there exists a probability space $\mathcal{P}_{s'}$ such that $s' \overset{a\restriction \mathit{ext}(M_2)}{\Longrightarrow}_{\mathrm{C}} \mathcal{P}_{s'}$, and
   (b) there exists a probability space $\mathcal{P}'_2$ of $\mathit{Probs}(\mathit{Probs}(\mathit{states}(M_2)))$ satisfying $\mathcal{P}_1 \sqsubseteq_{\mathcal{R}} \mathcal{P}'_2$,

   such that $\sum_{s' \in \Omega'} P'[s']\mathcal{P}_{s'} = \sum_{\mathcal{P} \in \Omega'_2} P'_2[\mathcal{P}]\mathcal{P}$.

We write $M_1 \sqsubseteq_{FS} M_2$ whenever $\mathit{ext}(M_1) = \mathit{ext}(M_2)$ and there is a probabilistic forward simulation from M_1 to M_2.

Example 8.5.2 (A probabilistic forward simulation) The probabilistic forward simulation for the probabilistic automata M_1 and M_2 of Figure 8-5 is the following: s_0 is related to $\mathcal{U}(s'_0)$; each state s_i, $i \geq 7$, is related to $\mathcal{D}(s'_i)$; each state s_i, $1 \leq i \leq 6$, is related to $\mathcal{U}(s'_{2i+1}, s'_{2i+2})$. It is an easy exercise to check that this relation is a probabilistic forward simulation. Observe also that there is no probabilistic forward simulation from M_2 to M_1. Informally, s'_3 cannot be simulated by M_1, since the only candidate state to be related to s'_1 is s_1, and s_1 does not contain all the information contained in s'_3. The formal way to see that there is no probabilistic forward simulation from M_2 to M_1 is to observe that M_2 and M_1 are not in the trace distribution precongruence relation and then use the fact that probabilistic forward simulations are sound for the trace distribution precongruence relation (cf. Section 8.7). In $M_2 \| C_P$ it is possible force action *left* to be scheduled exactly when M_2 is in s'_3, and thus it is possible to create a correlation between action *left* and actions a and b; in $M_1 \| C_P$ such a correlation cannot be created since action *left* must be scheduled before action l. ■

It is easy to check that a weak probabilistic simulation is a special case of a probabilistic forward simulation where each state of M_1 is related to a Dirac distribution. The verification that $\sqsubseteq_{FS}$

is a preorder that is preserved by parallel composition is more complicated. In this section we show that $\sqsubseteq_{FS}$ is preserved by parallel composition; the proof that $\sqsubseteq_{FS}$ is a preorder is postponed to Section 8.6.4.

Proposition 8.5.1 *$\sqsubseteq_{FS}$ is preserved by the parallel composition operator.*

Proof. Let $M_1 \sqsubseteq_{FS} M_2$, and let $\mathcal{R}$ be a probabilistic forward simulation from M_1 to M_2. Let $\mathcal{R}'$ be a relation between $\mathit{states}(M_1) \times \mathit{states}(M_3)$ and $\mathit{Probs}(\mathit{states}(M_2) \times \mathit{states}(M_3))$, defined as follows:

$$(s_1, s_3)\, \mathcal{R}'\, \mathcal{P} \text{ iff } \mathcal{P} = \mathcal{P}_2 \otimes \mathcal{D}(s_3) \text{ for some } \mathcal{P}_2 \text{ such that } s_1\, \mathcal{R}\, \mathcal{P}_2. \tag{8.4}$$

Condition 1 of the definition of a probabilistic forward simulation is immediate to verify. Condition 2 for transitions that involve M_1 only or M_3 only is immediate to verify as well.

Let $(s_1, s_3)\ \mathcal{R}'\ \mathcal{P}_2 \otimes \mathcal{D}(s_3)$, and let $(s_1, s_3) \xrightarrow{a} \mathcal{P}_1 \otimes \mathcal{P}_3$, where $s_1 \xrightarrow{a} \mathcal{P}_1$, and $s_3 \xrightarrow{a} \mathcal{P}_3$. From the definition of a probabilistic forward simulation, for each $s \in \Omega_2$ there exists a weak combined transition $s_2 \overset{a}{\Longrightarrow}_C \mathcal{P}_s$ of M_2, and there exists a probability space $\mathcal{P}_2'$ of $\mathit{Probs}(\mathit{Probs}(\mathit{states}(M_2)))$, such that

$$\sum_{s \in \Omega_2} P_2[s]\mathcal{P}_s = \sum_{\mathcal{P} \in \Omega_2'} P_2'[\mathcal{P}]\mathcal{P}, \tag{8.5}$$

and

$$\mathcal{P}_1 \sqsubseteq_{\mathcal{R}} \mathcal{P}_2'. \tag{8.6}$$

For each $s \in \Omega_2$, let $\mathcal{O}_s$ be a generator for $s \overset{a}{\Longrightarrow}_C \mathcal{P}_s$. Define a new generator $\mathcal{O}_s'$ as follows: for each finite execution fragment α of $M_2 \| M_3$ starting in (s, s_3),

1. if $\mathcal{O}_s(\alpha\lceil M_2) = (s', \mathcal{P})$, where $(s', \mathcal{P}) = \sum_i p_i(s', a_i, \mathcal{P}_i)$, each $(s', a_i, \mathcal{P}_i)$ is a transition of M_2, and $\alpha\lceil M_3 = s_3$, then

   $$\mathcal{O}_s'(\alpha) = \sum_i p_i((s', s_3), a_i, \mathcal{P}_i \otimes \mathcal{P}_i'),$$

   where

   $$\mathcal{P}_i' = \mathcal{D}(s_3) \text{ if } a_i \neq a, \text{ and } \mathcal{P}_i' = \mathcal{P}_3 \text{ if } a_i = a.$$

2. if $\mathcal{O}_s(\alpha\lceil M_2) = (s', \mathcal{P})$, where $(s', \mathcal{P}) = \sum_i p_i(s', a_i, \mathcal{P}_i)$, each $(s', a_i, \mathcal{P}_i)$ is a transition of M_2, $\alpha\lceil M_3 = s_3 a s_3'$, and $s_3' \in \Omega_3$, then

   $$\mathcal{O}_s'(\alpha) = \sum_i p_i((s', s_3'), a_i, \mathcal{P}_i \otimes \mathcal{D}(s_3'));$$

3. if none of the above cases holds, then $\mathcal{O}_s'(\alpha) = \mathcal{D}(\delta)$.

The weak combined transition generated by each $\mathcal{O}'_s$ is $(s, s_3) \overset{a}{\Longrightarrow}_{\mathrm{C}} \mathcal{P}_s \otimes \mathcal{P}_3$. In fact, an execution fragment α of $M_2 \| M_3$ is terminal for $\mathcal{O}'_s$ iff $\alpha \lceil M_2$ is terminal for $\mathcal{O}_s$ and $\alpha \lceil M_3 = s_3 a s'_3$ for $s'_3 \in \Omega_3$, and thus $\Omega_{\mathcal{O}'_s} = \Omega_s \times \Omega_3$. Moreover, for each $\alpha \in \Omega_{\mathcal{O}'_s}$, $P_\alpha^{\mathcal{O}'_s} = P_{\alpha \lceil M_2}^{\mathcal{O}_s} P_3[lstate(\alpha \lceil M_3)]$.

Denote $\mathcal{P}_s \otimes \mathcal{P}_3$ by $\mathcal{P}_{(s,s_3)}$. Then, for each $(s, s_3) \in \Omega_2 \times \mathcal{D}(s_3)$, we have identified a weak combined transition $(s, s_3) \overset{a}{\Longrightarrow}_{\mathrm{C}} \mathcal{P}_{(s,s_3)}$. These are the spaces of Condition 2.a in the definition of a probabilistic forward simulation. Note that $\mathcal{P}_{(s,s_3)}$ can be expressed alternatively as

$$\mathcal{P}_{(s,s_3)} = \sum_{s'_3 \in \Omega_3} P_3[s'_3] \left(\mathcal{P}_s \otimes \mathcal{D}(s'_3)\right). \tag{8.7}$$

Let

$$\mathcal{P}'_{2,3} \triangleq \sum_{s'_3 \in \Omega_3} P_3[s'_3] \left(\mathcal{P}'_2 \otimes \mathcal{D}(\mathcal{D}(s'_3))\right), \tag{8.8}$$

where the pairing of two probability spaces is meant to be their product. For each $s'_3 \in \Omega_3$, since $\mathcal{P}_1 \sqsubseteq_{\mathcal{R}} \mathcal{P}'_2$, $\mathcal{P}_1 \otimes \mathcal{D}(s'_3) \sqsubseteq_{\mathcal{R}} \mathcal{P}'_2 \otimes \mathcal{D}(\mathcal{D}(s'_3))$. Thus, from Lemma 8.2.1, $\mathcal{P}_1 \otimes \mathcal{P}_3 \sqsubseteq_{\mathcal{R}} \mathcal{P}'_{2,3}$. This is enough to show that Condition 2.b of the definition of a probabilistic forward simulation is satisfied.

We are left with $\sum_{s \in \Omega_2} P_2[s] \mathcal{P}_{(s,s_3)} = \sum_{\mathcal{P} \in \Omega'_{2,3}} P'_{2,3}[\mathcal{P}] \mathcal{P}$, which is shown as follows. From (8.7),

$$\sum_{s \in \Omega_2} P_2[s] \mathcal{P}_{(s,s_3)} = \sum_{s \in \Omega_2} \sum_{s'_3 \in \Omega_3} P_2[s] P_3[s'_3] \left(\mathcal{P}_s \otimes \mathcal{D}(s'_3)\right). \tag{8.9}$$

From (8.5),

$$\sum_{s \in \Omega_2} P_2[s] \mathcal{P}_{(s,s_3)} = \sum_{s'_3 \in \Omega_3} \sum_{\mathcal{P} \in \Omega'_2} P'_2[\mathcal{P}] P_3[s'_3] \left(\mathcal{P} \otimes \mathcal{D}(s'_3)\right). \tag{8.10}$$

From a simple algebraic manipulation,

$$\sum_{s \in \Omega_2} P_2[s] \mathcal{P}_{(s,s_3)} = \sum_{s'_3 \in \Omega_3} \sum_{\mathcal{P} \in \Omega_{\mathcal{P}'_2 \otimes \mathcal{D}(\mathcal{D}(s'_3))}} P_3[s'_3] P'_2[\mathcal{P}] \mathcal{P}. \tag{8.11}$$

From (8.8),

$$\sum_{s \in \Omega_2} P_2[s] \mathcal{P}_{(s,s_3)} = \sum_{\mathcal{P} \in \Omega'_{2,3}} P'_{2,3}[\mathcal{P}] \mathcal{P}. \tag{8.12}$$

■

8.6 The Execution Correspondence Theorem

The existence of some simulation relation between two probabilistic automata implies that there is some strict relation between their probabilistic executions. This relationship is known as the *execution correspondence lemma* for ordinary automata [GSSL94] and is useful in the context of liveness. In this section we prove the execution correspondence theorem for probabilistic automata; a corollary, which is proved in Section 8.7, is that the existence of a probabilistic forward simulation is sound for the trace distribution precongruence.

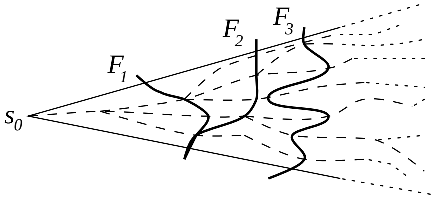


Figure 8-6: Fringes.

8.6.1 Fringes

Let H be a probabilistic execution of a probabilistic automaton M. Define the extended states of H, denoted by *extstates*(H), to be *states*$(H) \cup \{q\delta \mid q \in$ *states*$(H), P_H[C_{q\delta}] > 0\}$. A *fringe* of H is a discrete probability space $\mathcal{P}$ of *Probs*(*extstates*(H)) such that for each state q of H,

$$\sum_{q' \in \Omega | q \leq q'} P[q'] \leq P_H[C_q]. \tag{8.13}$$

Two fringes $\mathcal{P}_1$ and $\mathcal{P}_2$ are in the $\leq$ relation iff for each state q of H,

$$\sum_{q' \in \Omega_1 | q \leq q'} P_1[q'] \leq \sum_{q' \in \Omega_2 | q \leq q'} P_2[q']. \tag{8.14}$$

Informally, a fringe is a line that cuts a probabilistic execution in two parts (see Figure 8-6). A fringe is smaller than another one if the first fringe cuts the probabilistic execution earlier than the second fringe. Figure 8-6 shows three fringes F_1, F_2 and F_3, where $F_1 \leq F_2 \leq F_3$.

A fringe of particular interest is the fringe that cuts a probabilistic execution fragment at some depth i. Let *fringe*(H, i) denote the fringe of H where $\Omega = \{q \in$ *extstates*$(H) \mid |q| = i\} \cup \{q\delta \in$ *extstates*$(H) \mid |q| < i\}$, and for each $q \in \Omega$, $P[q] = P_H[C_q]$.

8.6.2 Execution Correspondence Structure

Let $\mathcal{R}$ be a probabilistic forward simulation from M_1 to M_2. An *execution correspondence structure* via $\mathcal{R}$ is a tuple (H_1, H_2, m, S), where H_1 is a probabilistic execution of M_1, H_2 is a probabilistic execution of M_2, m is a mapping from natural numbers to fringes of M_2, and S is a mapping from natural numbers to probability distributions of *Probs*(*Probs*(*states*(H_2))), such that

1. For each i, $m(i) \leq m(i+1)$;
2. For each state q_2 of H_2, $\lim_{i \to \infty} \sum_{q \in \Omega_i | q_2 \leq q} P_i[q] = P_H[C_q]$;
3. Let $q_1 \; \mathcal{R} \; \mathcal{P}$ iff for each $q \in \Omega$, *trace*$(q) =$ *trace*(q_1), and either
   (a) q_1 does not end in δ, each state of Ω does not end in δ, and *lstate*$(q_1) \; \mathcal{R}$ *lstate*$(\mathcal{P})$, or
   (b) q_1 and each state of Ω end in δ and *lstate*(δ-*strip*$(q_1)) \; \mathcal{R}$ *lstate*(δ-*strip*$(\mathcal{P})$).

   Then, for each $i \geq 0$, $m(i) = \sum_{\mathcal{P} \in \Omega_{S(i)}} P_{S(i)}[\mathcal{P}]\mathcal{P}$, and *fringe*$(H_1, i) \sqsubseteq_{\mathcal{R}} S(i)$.

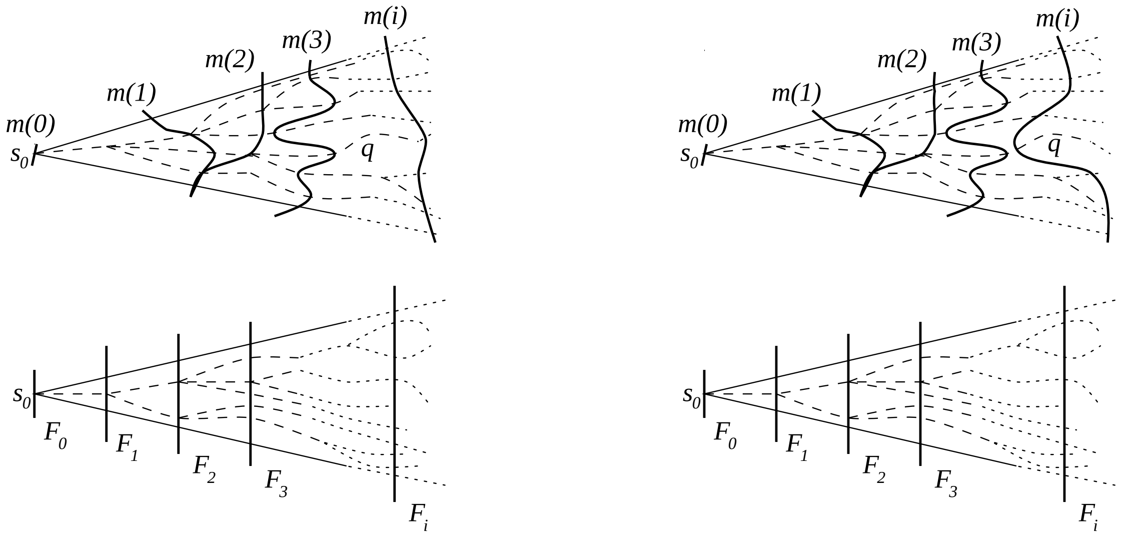


Figure 8-7: Execution Correspondence Structures: the role of Condition 2.

4. Let, for each $i \geq 0$, each $q_1 \in \mathit{fringe}(H_1, i)$, and each $q_2 \in \mathit{states}(H_2)$, $W_i(q_1, q_2) \triangleq \sum_{\mathcal{P}} w_i(q_1, \mathcal{P})P[q_2]$. If $W_i(q_1, q_2') = 0$ for each prefix or extension q_2' of q_2, then, for each extension q_1' of q_1 such that $q_1' \in \mathit{fringe}(H_1, i+1)$ and each prefix or extension q_2' of q_2, $W_{i+1}(q_1', q_2') = 0$.

Informally, an execution correspondence structure is an object that shows how a probabilistic execution H_1 of M_1 is represented by a probabilistic execution H_2 of M_2 via $\mathcal{R}$. H_2 is said to be the probabilistic execution fragment that corresponds to H_1. Conditions 1 and 3 state that each fringe $\mathit{fringe}(H_1, i)$ is represented by the fringe $m(i)$ in H_2, and Condition 2 states that at the limit each state of H_2 represents some part of H_1. Figure 8-7 gives an example of an execution correspondence structure (left) and of a structure that fails to satisfy Condition 2 since state q is not captured (right). Condition 4 enforces the correspondence between H_1 and H_2. Informally, it states that if two states q_1 and q_2 of H_1 and H_2, respectively, are connected through the i^{th} fringes, then for each $j < i$ there are two prefixes q_1' and q_2' of q_1 and q_2, respectively, that are connected through the j^{th} fringes. This condition allows us to derive a correspondence structure between the execution fragments of M_1 and M_2 that denote the states of H_1 and H_2. We do not use Condition 4 to prove any of the results that we present in this thesis; however, this condition is necessary to prove the results that Segala and Lynch present in [SL94].

If $\mathcal{R}$ is a weak probabilistic simulation, then an execution correspondence structure is a triplet (H_1, H_2, m): Condition 3 becomes $\mathit{fringe}(H_1, i) \sqsubseteq_{\mathcal{R}} m(i)$, where $q_1 \mathrel{\mathcal{R}} q_2$ iff $\mathit{trace}(q_1) = \mathit{trace}(q_2)$ and either q_1 and q_2 end in δ and $\delta\text{-}\mathit{strip}(\mathit{lstate}(q_1)) \mathrel{\mathcal{R}} \delta\text{-}\mathit{strip}(\mathit{lstate}(q_2))$, or $\mathit{lstate}(q_1) \mathrel{\mathcal{R}} \mathit{lstate}(q_2)$; $W_i(q_1, q_2)$ becomes $w_i(q_1, q_2)$, and Condition 4 says that for each $i \geq 0$, given $q_1 \in \mathit{fringe}(H_1, i)$ and $q_2 \in \mathit{states}(H_2)$, if $w_i(q_1, q_2') = 0$ for each prefix or extension q_2' of q_2, then, for each extension q_1' of q_1 such that $q_1' \in \mathit{fringe}(H_1, i+1)$, and each prefix or extension q_2' of q_2, $w_{i+1}(q_1', q_2') = 0$.

If $\mathcal{R}$ is a strong probabilistic simulation, then an execution correspondence structure is a pair (H_1, H_2): Conditions 1 and 2 are removed; Condition 3 becomes $\mathit{fringe}(H_1, i) \sqsubseteq_{\mathcal{R}} \mathit{fringe}(H_2, i)$ where $q_1 \mathcal{R} q_2$ iff $\mathit{itrace}(q_1) = \mathit{itrace}(q_2)$ and either q_1 and q_2 end in δ and $\delta\text{-}\mathit{strip}(\mathit{lstate}(q_1)) \mathcal{R}$ $\delta\text{-}\mathit{strip}(\mathit{lstate}(q_2))$, or $\mathit{lstate}(q_1) \mathcal{R} \mathit{lstate}(q_2)$; Condition 4 says that for each $i \geq 0$, given $q_1 \in \mathit{fringe}(H_1, i)$ and $q_2 \in \mathit{fringe}(H_2, i)$, if $w_i(q_1, q_2) = 0$, then, for each extension q_1' of q_1 such that $q_1' \in \mathit{fringe}(H_1, i{+}1)$ and each extension q_2' of q_2 such that $q_2' \in \mathit{fringe}(H_2, i{+}1)$, $w_{i+1}(q_1', q_2') = 0$.

8.6.3 The Main Theorem

Theorem 8.6.1 *Let $M_1 \sqsubseteq_{FS} M_2$ via the probabilistic forward simulation $\mathcal{R}$, and let H_1 be a probabilistic execution of M_1. Then there exists a probabilistic execution H_2 of M_2, a mapping m from natural numbers to fringes of M_2, and a mapping S from natural numbers to probability distributions of $\mathit{Probs}(\mathit{Probs}(\mathit{states}(H_2)))$, such that (H_1, H_2, m, S) is an execution correspondence structure via $\mathcal{R}$.*

Proof. Let q_1 be a state of H_1, and let $\mathcal{P}_2$ be a distribution over potential states of H_2 such that $q_1 \sqsubseteq_{\mathcal{R}} \mathcal{P}_2$ according to the definition given in the definition of an execution correspondence structure. Denote by $\mathcal{P}_{H_1}^{q_1}$ the probability space such that $tr_{q_1}^{H_1} = \sum_{tr \in \Omega_{H_1}^{q_1}} P_{H_1}^{q_1}[tr](q_1 \frown tr)$. Let $tr_1 \in \Omega_{H_1}^{q_1}$, and let $\mathcal{P}_{tr_1}$ be the probability space reached in $q_1 \frown tr_1$.

Since $\mathcal{R}$ is a probabilistic forward simulation, then for each state q_2 of Ω_2 there exists a weak transition $tr_{q_1 \mathcal{P}_2 tr_1 q_2}$ of H_2 with action $a \restriction \mathit{ext}(M_2)$, leading to a distribution over states $\mathcal{P}_{q_1 \mathcal{P}_2 tr_1 q_2}$, such that there exists a probability distribution over probability distributions of potential states of H_2, denoted by $\mathcal{P}_{q_1 \mathcal{P}_2 tr_1}^S$, satisfying

$$\sum_{\mathcal{P} \in \Omega_{q_1 \mathcal{P}_2 tr_1}^S} P_{q_1 \mathcal{P}_2 tr_1}^S[\mathcal{P}]\mathcal{P} = \sum_{q_2 \in \Omega_2} P_2[q_2]\mathcal{P}_{q_1 \mathcal{P}_2 tr_1 q_2} \tag{8.15}$$

and

$$\mathcal{P}_{tr_1} \sqsubseteq_{\mathcal{R}} \mathcal{P}_{q_1 \mathcal{P}_2 tr_1}^S \tag{8.16}$$

via a weight function $w_{q_1 \mathcal{P}_2 tr_1}$. Denote the probability space $\sum_{q_2 \in \Omega_2} P_2[q_2]\mathcal{P}_{q_1 \mathcal{P}_2 tr_1 q_2}$ by $\mathcal{P}_{q_1 \mathcal{P}_2 tr_1}$, i.e.,

$$\mathcal{P}_{q_1 \mathcal{P}_2 tr_1} \triangleq \sum_{q_2 \in \Omega_2} P_2[q_2]\mathcal{P}_{q_1 \mathcal{P}_2 tr_1 q_2}. \tag{8.17}$$

Denote the generator of each weak transition $tr_{q_1 \mathcal{P}_2 tr_1 q_2}$ by $\mathcal{O}_{q_1 \mathcal{P}_2 tr_1 q_2}$ (cf. Section 4.2.7). For the sake of this proof, we change the notation for the generators of the transitions of a probabilistic execution. Thus, for each q_2' such that $q_2 \leq q_2'$, $\mathcal{O}_{q_1 \mathcal{P}_2 tr_1 q_2}(q_2')$ stands for $\mathcal{O}_{q_1 \mathcal{P}_2 tr_1 q_2}(q_2' \uparrow q_2)$, and $P_{q_2'}^{\mathcal{O}_{q_1 \mathcal{P}_2 tr_1 q_2}}$ stands for $P_{q_2' \uparrow q_2}^{\mathcal{O}_{q_1 \mathcal{P}_2 tr_1 q_2}}$.

For each state q_1 and each probability distribution over states $\mathcal{P}_2$, let $\delta_{q_1} \triangleq \mathcal{D}(q_1 \delta)$, $\delta_{\mathcal{P}_2} \triangleq \sum_{q_2 \in \Omega_2} P_2[q_2]\delta_{q_2}$, $\delta_{\mathcal{P}_2}^S \triangleq \mathcal{D}(\delta_{\mathcal{P}_2})$, and $w_{\delta q_1 \mathcal{P}_2}$ be a weight function such that $w_{\delta q_1 \mathcal{P}_2}(q_1 \delta, \mathcal{P}_2) = 1$. Note that, if for each $q_2 \in \Omega_2$, $\mathit{trace}(q_1) = \mathit{trace}(q_2)$, then

$$\delta_{q_1} \sqsubseteq_{\mathcal{R}} \delta_{\mathcal{P}_2}^S \tag{8.18}$$

via $w_{\delta q_1 \mathcal{P}_2}$. Moreover,

$$\delta_{\mathcal{P}_2} = \sum_{\mathcal{P} \in \Omega^S_{\mathcal{P}_2}} P_{\delta^S_{\mathcal{P}_2}}[\mathcal{P}]\mathcal{P}. \tag{8.19}$$

Let s_1 be the start state of H_1, and s_2 be a start state of M_2 that is related to s_1. We know that s_2 exists since $\mathcal{R}$ is a probabilistic forward simulation. Let *Active* be the smallest set such that

1. $(s_1, \mathcal{D}(s_2)) \in \mathit{Active}$;
2. if $(q_1, \mathcal{P}_2) \in \mathit{Active}$, $tr_1 \in \Omega^{q_1}_{H_1}$, and $(q_1', \mathcal{P}_2') \in \Omega_{tr_1} \times \Omega^S_{q_1 \mathcal{P}_2 tr_1}$, then $(q_1', \mathcal{P}_2') \in \mathit{Active}$;
3. if $(q_1, \mathcal{P}_2) \in \mathit{Active}$, $P^{q_1}_{H_1}[\delta] > 0$, then $(q_1\delta, \delta^S_{\mathcal{P}_2}) \in \mathit{Active}$.

Observe that for each pair $(q_1, \mathcal{P}_2) \in \mathit{Active}$, $q_1 \mathrel{\mathcal{R}} \mathcal{P}_2$ (simple inductive argument). For each q_1 such that there exists some $\mathcal{P}_2$ with $(q_1, \mathcal{P}_2) \in \mathit{Active}$, each $tr_1 \in \Omega^{q_1}_{H_1}$, and each $q_2 \in \Omega_2$, let $\mathit{active}(q_1, \mathcal{P}_2, tr_1, q_2)$ be the set of states that are active in $\mathcal{O}_{q_1 \mathcal{P}_2 tr_1 q_2}$, and let $\mathit{reach}(q_1, \mathcal{P}_2, tr_1, q_2)$ be the set of states that are reachable in $\mathcal{O}_{q_1 \mathcal{P}_2 tr_1 q_2}$. Let *active* denote the union of the sets $\mathit{reach}(q_1, \mathcal{P}_2, tr_1, q_2)$ where $(q_1, \mathcal{P}_2) \in \mathit{Active}$, $tr_1 \in \Omega^{q_1}_{H_1}$, and $q_2 \in \Omega_2$. For each $i \leq 0$, let $\mathit{Active}(i)$ be the set of pairs $(q_1, \mathcal{P}_2) \in \mathit{Active}$ such that either $|q_1| = i$ or $|q_1| \leq i$ and q_1 ends in δ. For each pair $(q_1, \mathcal{P}_2)$ of *Active* such that q_1 does not end in δ, let

$$\mathcal{P}_{q_1} \triangleq \sum_{tr_1 \in \Omega^{q_1}_{H_1}} P^{q_1}_{H_1}[tr_1]\mathcal{P}_{tr_1} + P^{q_1}_{H_1}[\delta]\delta_{q_1} \tag{8.20}$$

be the probability space reached in H_1 with the transition enabled from q_1,

$$\mathcal{P}_{q_1 \mathcal{P}_2} \triangleq \sum_{tr_1 \in \Omega^{q_1}_{H_1}} P^{q_1}_{H_1}[tr_1]\mathcal{P}_{q_1 \mathcal{P}_2 tr_1} + P^{q_1}_{H_1}[\delta]\delta_{\mathcal{P}_2} \tag{8.21}$$

be the probability space that is reached in the corresponding transition of $\mathcal{P}_2$,

$$\mathcal{P}^S_{q_1 \mathcal{P}_2} \triangleq \sum_{tr_1 \in \Omega^{q_1}_{H_1}} P^{q_1}_{H_1}[tr_1]\mathcal{P}^S_{q_1 \mathcal{P}_2 tr_1} + P^{q_1}_{H_1}[\delta]\delta^S_{\mathcal{P}_2} \tag{8.22}$$

be the probability space of probability spaces that corresponds to $\mathcal{P}_{q_1}$, and for each $q_1', \mathcal{P}_2'$,

$$w_{q_1 \mathcal{P}_2}(q_1', \mathcal{P}_2') \triangleq \sum_{tr_1 \in \Omega^{q_1}_{H_1}} P^{q_1}_{H_1}[tr_1] w_{q_1 \mathcal{P}_2 tr_1}(q_1', \mathcal{P}_2') + P^{q_1}_{H_1}[\delta] w_{\delta q_1 \mathcal{P}_2}(q_1', \mathcal{P}_2') \tag{8.23}$$

be the corresponding weight function. From Lemma 8.2.1,

$$\mathcal{P}_{q_1} \sqsubseteq_{\mathcal{R}} \mathcal{P}^S_{q_1 \mathcal{P}_2} \tag{8.24}$$

via the weight function $w_{q_1 \mathcal{P}_2}$.
For each pair $(q_1, \mathcal{P}_2)$ of *Active* such that q_1 ends in δ, let

$$\mathcal{P}_{q_1} \triangleq \mathcal{D}(q_1), \quad \mathcal{P}_{q_1, \mathcal{P}_2} \triangleq \mathcal{P}_2, \quad \mathcal{P}^S_{q_1, \mathcal{P}_2} \triangleq \mathcal{D}(\mathcal{P}_2), \quad \text{and } w_{q_1 \mathcal{P}_2}(q_1, \mathcal{P}_2) \triangleq 1. \tag{8.25}$$

It is immediate to observe that Equation (8.24) holds also in this case.

Define $m(i), S(i)$ and w_i inductively as follows.

$$m(0) \triangleq \mathcal{D}(s_2), \quad S(0) \triangleq \mathcal{D}(m(0)), \quad w_0(s_1, m(0)) \triangleq 1, \tag{8.26}$$

$$\begin{aligned}
m(i+1) &\triangleq \sum_{(q_1,\mathcal{P}_2)\in Active(i)} w_i(q_1,\mathcal{P}_2)\mathcal{P}_{q_1\mathcal{P}_2}, && (8.27)\\
S(i+1) &\triangleq \sum_{(q_1,\mathcal{P}_2)\in Active(i)} w_i(q_1,\mathcal{P}_2)\mathcal{P}^S_{q_1\mathcal{P}_2}, && (8.28)\\
w_{i+1}(q_1',\mathcal{P}_2') &\triangleq \sum_{(q_1,\mathcal{P}_2)\in Active(i)} w_i(q_1,\mathcal{P}_2)w_{q_1\mathcal{P}_2}(q_1',\mathcal{P}_2'). && (8.29)
\end{aligned}$$

To show that Equations (8.27), (8.28),and (8.29) are well defined, we show by induction that for each $i \geq 0$, $\sum_{(q_1,\mathcal{P}_2)\in Active(i)} w_i(q_1,\mathcal{P}_2) = 1$. The base case is a direct consequence of (8.26) and the definition of $Active(0)$. For the inductive step,

$$\begin{aligned}
&\sum_{(q_1,\mathcal{P}_2)\in Active(i+1)} w_{i+1}(q_1,\mathcal{P}_2)\\
&= \sum_{(q_1,\mathcal{P}_2)\in Active(i+1)} \sum_{(q_1',\mathcal{P}_2')\in Active(i)} w_i(q_1',\mathcal{P}_2')w_{q_1'\mathcal{P}_2'}(q_1,\mathcal{P}_2)\\
&= \sum_{(q_1',\mathcal{P}_2')\in Active(i)} w_i(q_1',\mathcal{P}_2')\\
&= 1,
\end{aligned}$$

where the first step follows from Equation (8.29), the second step follows from the fact that $w_{q_1',\mathcal{P}_2'}$ is a weight function that is non zero only in pairs of $Active(i+1)$, and the third step follows from induction. Let

$$W_{q_1\mathcal{P}_2tr_1q_2}(q_2') \triangleq w(q_1,\mathcal{P}_2)P_{H_1}^{q_1}[tr_1]P_2[q_2]P_{q_2'}^{\mathcal{O}_{q_1\mathcal{P}_2tr_1q_2}}. \tag{8.30}$$

Consider a state q_2 of *active*. Then the transition enabled from q_2 is

$$\sum_{(q_1',\mathcal{P}_2')\in Active} \sum_{tr_1\in\Omega_{q_1'}^{H_1}} \sum_{q_2'\in\Omega_2'|q_2\in active(q_1',\mathcal{P}_2',tr_1,q_2')} \tag{8.31}$$

$$P_{\mathcal{O}_{q_1\mathcal{P}_2tr_1q_2'}(q_2)}[acts(M_2)]W_{q_1'\mathcal{P}_2'tr_1q_2'}(q_2)/W(q_2)\left(\mathcal{O}_{q_1'\mathcal{P}_2tr_1q_2'}(q_2)\restriction acts(M_2)\right),$$

where $W(s_2) \triangleq 1$, and for each $q_2 \neq s_2$,

$$W(q_2) \triangleq \sum_{(q_1',\mathcal{P}_2')\in Active} \sum_{tr_1\in\Omega_{q_1'}^{H_1}} \sum_{q_2'\in\Omega_2'|q_2'\neq q_2, q_2\in reach(q_1',\mathcal{P}_2',tr_1,q_2')} W_{q_1'\mathcal{P}_2'tr_1q_2'}(q_2). \tag{8.32}$$

It is easy to verify that Expression (8.31) denotes a valid transition of a probabilistic execution fragment of M since it is the combination of legal transitions of a probabilistic execution fragment of M. The fact that the projection of a legal transition of a probabilistic execution fragment of M onto $acts(M)$ is still a legal transition of a probabilistic execution fragment of M follows from the fact that M is a simple probabilistic automaton.

Informally, the set *active* is used to identify all the states of H_2. The transition enabled from each one of those states, say q_2, is due to several states of H_1, and each state of H_1 influences the transition enabled from a specific state of H_2 with a different probability. Such a probability depends on how much a state of H_2 represents a state of H_1, on the probability of the transition of M_1 that has to be matched, on the probability of reaching a specific state q_2' of H_2 during the matching operation, on the probability of reaching q_2 from q_2', and on the probability of departing from q_2. These conditions are captured by $P_{\mathcal{O}_{q_1 \mathcal{P}_2 tr_1 q_2'}(q_2)}[acts(M_2)] W_{q_1' \mathcal{P}_2' tr_1 q_2'}(q_2)$. These weights must be normalized with respect to the probability of reaching q_2, which is expressed by $W(q_2)$. The condition $q_2' \neq q_2$ in the third sum of (8.32) is justified by the fact $W(q_2)$ is the probability of reaching q_2.

This completes the definition of H_2, $m(i)$, $S(i)$, and the w_i's. We need to show that (H_1, H_2, w, S) is an execution correspondence structure via $\mathcal{R}$. Thus, we need to show the following properties.

1. For each i, $m(i)$ is a fringe of H_2;
2. For each i, $m(i) \leq m(i+1)$;
3. For each state q of H_2, $\lim_{i\to\infty} \sum_{q' \in \Omega_i | q \leq q'} P_i[q'] = P_H[C_q]$;
4. For each i, $m(i) = \sum_{\mathcal{P} \in S(i)} P_{S(i)}[\mathcal{P}]\mathcal{P}$;
5. For each i, $\mathit{fringe}(H_1, i) \sqsubseteq_{\mathcal{R}} S(i)$ via w_i.
6. For each i, each $q_1 \in \mathit{fringe}(H_1, i)$, and each $q_2 \in \mathit{states}(H_2)$, if $W_i(q_1, q_2') = 0$ for each prefix or extension q_2' of q_2, then, for each extension q_1' of q_1 such that $q_1' \in \mathit{fringe}(H_1, i+1)$ and each prefix or extension q_2' of q_2, $W_{i+1}(q_1', q_2') = 0$.

We show each item separately.

1. For each i, $m(i)$ is a fringe of H_2.

   By construction $m(i)$ is a probability distribution. Thus, we need to show only that for each state q_2 of H_2,

$$\sum_{q_2' \in \Omega_{m(i)} | q_2 \leq q_2'} P_{m(i)}[q_2'] \leq P_{H_2}[C_{q_2}] \tag{8.33}$$

   First we show that for each $q_2 \in \mathit{states}(H_2)$,

$$W(q_2) = P_{H_2}[C_{q_2}]; \tag{8.34}$$

   then we show that for each $q_2 \in \mathit{states}(H_2)$,

$$\sum_{q_2' \in \Omega_{m(i)} | q_2 \leq q_2'} P_{m(i)}[q_2'] \leq W(q_2). \tag{8.35}$$

   The proof of (8.34) is by induction on the length of q_2. If $q_2 = s_2$, then (8.34) holds by definition. Otherwise, let $\tilde{q}_2$ be q_2 without its last action and state, i.e., $q_2 = \tilde{q}_2 as$ for some action a and some state s. Then, from the definition of the probability of a cone, induction, Equation (8.31) and an algebraic simplification,

$$P_{H_2}[C_{q_2}] = \sum_{(q_1',\mathcal{P}_2')\in Active} \sum_{tr_1\in\Omega_{H_1}^{q_1'}} \sum_{q_2'\in\Omega_2' \mid \tilde{q}_2\in active(q_1',\mathcal{P}_2',tr_1,q_2')}$$
$$W_{q_1'\mathcal{P}_2'tr_1q_2'}(\tilde{q}_2)P_{\mathcal{O}_{q_1'\mathcal{P}_2tr_1q_2'}(\tilde{q}_2)}[q_2]. \tag{8.36}$$

From Equation (8.30) and the definition of $P_{q_2}^{\mathcal{O}_{q_1'\mathcal{P}_2'tr_1q_2'}}$ (cf. Section 4.2.7), we obtain

$$P_{H_2}[C_{q_2}] = \sum_{(q_1',\mathcal{P}_2')\in Active} \sum_{tr_1\in\Omega_{H_1}^{q_1'}} \sum_{q_2'\in\Omega_2' \mid \tilde{q}_2\in active(q_1',\mathcal{P}_2',tr_1,q_2')}$$
$$w(q_1', \mathcal{P}_2')P_{H_1}^{q_1'}[tr_1]P_2'[q_2']P_{q_2}^{\mathcal{O}_{q_1'\mathcal{P}_2'tr_1q_2'}}. \tag{8.37}$$

Observe that $q_2' \in \Omega_2'$ and $\tilde{q}_2 \in active(q_1', \mathcal{P}_2', tr_1, q_2')$ iff $q_2' \in \Omega_2'$, $q_2' \neq q_2$, and $q_2 \in reach(q_1', \mathcal{P}_2', tr_1, q_2')$. Thus, from Equation (8.31),

$$P_{H_2}[C_{q_2}] = \sum_{(q_1',\mathcal{P}_2')\in Active} \sum_{tr_1\in\Omega_{H_1}^{q_1'}} \sum_{q_2'\in\Omega_2' \mid q_2'\neq q_2, q_2\in reach(q_1',\mathcal{P}_2',tr_1,q_2')} W_{q_1'\mathcal{P}_2'tr_1q_2'}(q_2). \tag{8.38}$$

At this point Equation (8.32) is sufficient to conclude the validity of Equation (8.34).

The proof of Equation (8.35) is also by induction. If $i = 0$, then the result follows directly from the fact that a fringe is a probability distribution. Otherwise, let $N(q_1)$ be true iff q_1 does not end in δ. Then, from Equation (8.27),

$$\sum_{q_2'\in\Omega_{m(i+1)} \mid q_2\leq q_2'} P_{m(i+1)}[q_2'] \tag{8.39}$$

can be rewritten into

$$\sum_{q_2'\in\Omega_{m(i+1)} \mid q_2\leq q_2'} \sum_{(q_1,\mathcal{P}_2)\in Active(i)} w_i(q_1, \mathcal{P}_2)P_{q_1\mathcal{P}_2}[q_2']. \tag{8.40}$$

From the definition of $\mathcal{P}_{q_1,\mathcal{P}_2}$ (Equations (8.21) and (8.25)) and the definition of $\mathcal{P}_{q_1\mathcal{P}_2tr_1}$ (Equation (8.17)), Expression (8.40) can be rewritten into

$$\sum_{q_2'\in\Omega_{m(i+1)} \mid q_2\leq q_2'} \sum_{(q_1,\mathcal{P}_2)\in Active(i), N(q_1)} \sum_{tr_1\in\Omega_{H_1}^{q_1}} \sum_{q_2''\in\Omega_2} \tag{8.41}$$
$$w_i(q_1, \mathcal{P}_2)P_{H_1}^{q_1}[tr_1]P_2[q_2'']P_{q_1\mathcal{P}_2tr_1q_2''}[q_2']$$
$$+ \sum_{q_2'\delta\in\Omega_{m(i+1)} \mid q_2\leq q_2'} \sum_{(q_1,\mathcal{P}_2)\in Active(i), N(q_1)} w_i(q_1, \mathcal{P}_2)P_{H_1}^{q_1}[\delta]P_2[q_2']$$
$$+ \sum_{q_2'\delta\in\Omega_{m(i+1)} \mid q_2\leq q_2'} \sum_{(q_1\delta,\mathcal{P}_2)\in Active(i)} w_i(q_1\delta, \mathcal{P}_2)P_2[q_2'\delta].$$

By exchanging sums in Expression (8.41), we obtain

$$\begin{aligned}&\sum_{(q_1,\mathcal{P}_2)\in Active(i),N(q_1)}\ \sum_{tr_1\in\Omega_{H_1}^{q_1}}\ \sum_{q_2''\in\Omega_2}\ \sum_{q_2'\in\Omega_{m(i+1)}|q_2\leq q_2'}\\ &\qquad w_i(q_1,\mathcal{P}_2)P_{H_1}^{q_1}[tr_1]P_2[q_2'']P_{q_1\mathcal{P}_2 tr_1 q_2''}[q_2']\\ &+\sum_{(q_1,\mathcal{P}_2)\in Active(i),N(q_1)}\ \sum_{q_2'\delta\in\Omega_{m(i+1)}|q_2\leq q_2'} w_i(q_1,\mathcal{P}_2)P_{H_1}^{q_1}[\delta]P_2[q_2']\\ &+\sum_{(q_1\delta,\mathcal{P}_2)\in Active(i)}\ \sum_{q_2'\delta\in\Omega_{m(i+1)}|q_2\leq q_2'} w_i(q_1\delta,\mathcal{P}_2)P_2[q_2'\delta],\end{aligned}\tag{8.42}$$

where the first summand comes from the first summand of (8.22), the second summand comes from the second summand of (8.22), and the third summand comes from (8.25). Consider the first summand of Expression (8.42), and partition the states q_2'' of Ω_2 into those that include q_2 ($q_2 \leq q_2''$) and those that do not. In the first case, since from (8.27), (8.21), and (8.17), $\Omega_{q_1\mathcal{P}_2 tr_1 q_2''} \subseteq \Omega_{m(i+1)}$, and since each element q_2' of $\Omega_{q_1\mathcal{P}_2 tr_1 q_2''}$ satisfies $q_2 \leq q_2'$,

$$\sum_{q_2'\in\Omega_{m(i+1)}|q_2\leq q_2'} P_{q_1\mathcal{P}_2 tr_1 q_2''}[q_2'] = 1;\tag{8.43}$$

in the second case the same sum gives $P_{q_2}^{\mathcal{O}_{q_1\mathcal{P}_2 tr_1 q_2''}}$. Consider the second summand of Expression (8.42), and observe that, from (8.27), (8.21), and the definition of $\delta_{\mathcal{P}_2}$, $q_2'\delta \in \Omega_{m(i+1)}$, $q_2 \leq q_2'$, and $P_2[q_2'] > 0$ iff $q_2' \in \Omega_2$, $q_2 \leq q_2'$, and $P_2[q_2'] > 0$. Finally, consider the third summand of Expression (8.42), and observe that all the states of Ω_2 end with δ, and, from (8.27) and (8.21), $q_2'\delta \in \Omega_{m(i+1)}$, $q_2 \leq q_2'$, and $P_2[q_2'\delta] > 0$ iff $q_2'\delta \in \Omega_2$, $q_2 \leq q_2'\delta$, $P_2[q_2'\delta] > 0$. By combining the observations above, Expression (8.42) can be rewritten into

$$\begin{aligned}&\sum_{(q_1,\mathcal{P}_2)\in Active(i),N(q_1)}\ \sum_{tr_1\in\Omega_{H_1}^{q_1}} w_i(q_1,\mathcal{P}_2)P_{H_1}^{q_1}[tr_1]\\ &\qquad\left(\sum_{q_2''\in\Omega_2|q_2\leq q_2''} P_2[q_2''] + \sum_{q_2''\in\Omega_2|q_2''<q_2} P_2[q_2'']P_{q_2}^{\mathcal{O}_{q_1\mathcal{P}_2 tr_1 q_2''}}\right)\\ &+\sum_{(q_1,\mathcal{P}_2)\in Active(i),N(q_1)}\ \sum_{q_2''\in\Omega_2|q_2\leq q_2''} w_i(q_1,\mathcal{P}_2)P_{H_1}^{q_1}[\delta]P_2[q_2'']\\ &+\sum_{(q_1\delta,\mathcal{P}_2)\in Active(i)}\ \sum_{q_2''\in\Omega_2|q_2\leq q_2''} w_i(q_1\delta,\mathcal{P}_2)P_2[q_2''].\end{aligned}\tag{8.44}$$

By regrouping expressions and simplifying, we obtain

$$\begin{aligned}&\sum_{(q_1,\mathcal{P}_2)\in Active(i),N(q_1)}\ \sum_{tr_1\in\Omega_{H_1}^{q_1}}\ \sum_{q_2''\in\Omega_2|q_2\leq q_2''} w_i(q_1,\mathcal{P}_2)P_{H_1}^{q_1}[tr_1]P_2[q_2'']P_{q_2}^{\mathcal{O}_{q_1\mathcal{P}_2 tr_1 q_2''}}\\ &+\sum_{(q_1,\mathcal{P}_2)\in Active(i)}\ \sum_{q_2''\in\Omega_2|q_2\leq q_2''} w_i(q_1,\mathcal{P}_2)P_2[q_2''].\end{aligned}\tag{8.45}$$

Finally, from Equation (8.30), Expression (8.45) can be rewritten into

$$\sum_{(q_1,\mathcal{P}_2)\in Active(i),N(q_1)} \sum_{tr_1\in\Omega_{H_1}^{q_1}} \sum_{q_2''\in\Omega_2|q_2\leq q_2''} W_{q_1\mathcal{P}_2 tr_1 q_2''}(q_2) + \sum_{(q_1,\mathcal{P}_2)\in Active(i)} \sum_{q_2''\in\Omega_2|q_2\leq q_2''} w_i(q_1,\mathcal{P}_2)P_2[q_2'']. \tag{8.46}$$

We now analyze the second summand of Expression (8.46), and we show by induction on i that it is 0 if $i = 0$ and $q_2 \neq s_2$, it is 1 if $i = 0$ and $q_2 = s_2$, and it is

$$\sum_{j<i} \sum_{(q_1,\mathcal{P}_2)\in Active(j)} \sum_{tr_1\in\Omega_{H_1}^{q_1}} \sum_{q_2''\in\Omega_2|q_2''<q_2} W_{q_1\mathcal{P}_2 tr_1 q_2''}(q_2) \tag{8.47}$$

otherwise. For $i = 0$ the result is trivial. Otherwise, from Equation (8.29),

$$\sum_{(q_1,\mathcal{P}_2)\in Active(i+1)} \sum_{q_2''\in\Omega_2|q_2\leq q_2''} w_{i+1}(q_1,\mathcal{P}_2)P_2[q_2''] \tag{8.48}$$

can be rewritten into

$$\sum_{(q_1,\mathcal{P}_2)\in Active(i+1)} \sum_{(q_1',\mathcal{P}_2')\in Active(i)} \sum_{q_2''\in\Omega_2|q_2\leq q_2''} w_i(q_1',\mathcal{P}_2')w_{q_1'\mathcal{P}_2'}(q_1,\mathcal{P}_2)P_2[q_2'']. \tag{8.49}$$

From the definition of $w_{q_1'\mathcal{P}_2'}$ (Equations (8.23) and (8.25)), Expression (8.49) can be rewritten into

$$\begin{aligned} &\sum_{(q_1,\mathcal{P}_2)\in Active(i+1)} \sum_{(q_1',\mathcal{P}_2')\in Active(i),N(q_1')} \sum_{tr_1'\in\Omega_{H_1}^{q_1'}} \sum_{q_2''\in\Omega_2|q_2\leq q_2''} \\ &\quad w_i(q_1',\mathcal{P}_2')P_{H_1}^{q_1'}[tr_1']w_{q_1'\mathcal{P}_2'tr_1'}(q_1,\mathcal{P}_2))P_2[q_2''] \\ &+ \sum_{(q_1\delta,\mathcal{P}_2)\in Active(i+1)} \sum_{(q_1',\mathcal{P}_2')\in Active(i),N(q_1')} \sum_{q_2''\in\Omega_2|q_2\leq q_2''} \\ &\quad w_i(q_1',\mathcal{P}_2')P_{H_1}^{q_1'}[\delta]w_{\delta q_1'\mathcal{P}_2'}(q_1\delta,\mathcal{P}_2)P_2[q_2''] \\ &+ \sum_{(q_1'\delta,\mathcal{P}_2')\in Active(i)} \sum_{q_2''\in\Omega_2'|q_2\leq q_2''} w_i(q_1'\delta,\mathcal{P}_2')P_2[q_2'']. \end{aligned} \tag{8.50}$$

Observe that in the first summand of (8.50)

$$\begin{aligned} &\sum_{(q_1,\mathcal{P}_2)\in Active(i+1)} \sum_{q_2''\in\Omega_2|q_2\leq q_2''} w_{q_1'\mathcal{P}_2'tr_1'}(q_1,\mathcal{P}_2)P_2[q_2''] \\ &= \sum_{\mathcal{P}_2|\exists q_1,(q_1,\mathcal{P}_2)\in Active(i+1)} \sum_{q_2''\in\Omega_2|q_2\leq q_2''} P^S_{q_1'\mathcal{P}_2'tr_1'}[\mathcal{P}_2]P_2[q_2''] \\ &= \sum_{q_2'''\in\Omega_2'} \sum_{q_2''\in\Omega_{q_1'\mathcal{P}_2'tr_1'}|q_2\leq q_2''} P_{q_1'\mathcal{P}_2'tr_1'q_2'''}[q_2''], \end{aligned}$$

where the first step follows from the fact that $w_{q'_1\mathcal{P}'_1 tr'_1 q'''_2}$ is a weight function, and the second step follows from (8.17), (8.15) and the fact that $\Omega_{q'_1\mathcal{P}'_2 tr'_1}$ is the set of probability space $\mathcal{P}_2$ such that there is a state q_1 where $(q_1, \mathcal{P}_2) \in \mathit{Active}(i+1)$ (cf. the definition of *Active* and observe that $|q_1| = i+1$). For the second summand of (8.50), observe that for each pair $(q'_1\delta, \mathcal{P}_2)$ of $\mathit{Active}(i+1)$, if $P^{q'_1}_{H_1}[\delta] > 0$, then there is exactly one pair $(q_1, \mathcal{P}'_2)$ of $\mathit{Active}(i)$ such that $w_{\delta q'_1 \mathcal{P}'_2}(q'_1\delta, \mathcal{P}_2) > 0$. In particular, $q_1 = q'_1$, $\mathcal{P}_2 = \delta_{\mathcal{P}'_2}$, and $w_{\delta q'_1 \mathcal{P}'_2}(q'_1\delta, \mathcal{P}_2) = 1$. Conversely, for each pair $(q'_1, \mathcal{P}'_2)$ of $\mathit{Active}(i)$ such that $P^{q'_1}_{H_1}[\delta] > 0$, the pair $(q'_1\delta, \mathcal{P}_2)$ is in $\mathit{Active}(i+1)$ and $w_{\delta q'_1 \mathcal{P}'_2}(q'_1\delta, \mathcal{P}_2) = 1$. Thus, the term $w_{\delta q'_1 \mathcal{P}'_2}(q'_1\delta, \mathcal{P}_2)$ and the sum $\sum_{(q'_1\delta, \mathcal{P}_2) \in \mathit{Active}(i+1)}$ can be removed from the second summand of (8.50). Thus, by applying the observations above to (8.50), we obtain

$$
\begin{aligned}
&\sum_{(q'_1,\mathcal{P}'_2)\in \mathit{Active}(i), N(q'_1)} \; \sum_{tr'_1 \in \Omega^{q'_1}_{H_1}} \; \sum_{q'''_2 \in \Omega'_2} \; \sum_{q''_2 \in \Omega_{q'_1\mathcal{P}'_2 tr'_1 q'''_2} | q_2 \le q''_2} \\
&\qquad w_i(q'_1, \mathcal{P}'_2) P^{q'_1}_{H_1}[tr'_1] P'_2[q'''_2] P_{q'_1\mathcal{P}'_2 tr'_1 q'''_2}[q''_2] \\
&+ \sum_{(q'_1,\mathcal{P}'_2)\in \mathit{Active}(i), N(q'_1)} \; \sum_{q'''_2 \in \Omega'_2 | q_2 \le q''_2} w_i(q'_1, \mathcal{P}'_2) P^{q'_1}_{H_1}[\delta] P'_2[q'''_2] \\
&+ \sum_{(q'_1\delta,\mathcal{P}'_2)\in \mathit{Active}(i)} \; \sum_{q'''_2 \in \Omega'_2 | q_2 \le q'''_2} w_i(q'_1\delta, \mathcal{P}'_2) P'_2[q'''_2].
\end{aligned}
\tag{8.51}
$$

Consider the first summand of Expression (8.51). If $q_2 \le q'''_2$, then

$$
\sum_{q''_2 \in \Omega_{q'_1\mathcal{P}'_2 tr'_1 q'''_2} | q_2 \le q''_2} P_{q'_1\mathcal{P}'_2 tr'_1 q'''_2}[q''_2] = 1;
\tag{8.52}
$$

If $q'''_2 \le q_2$, then

$$
\sum_{q''_2 \in \Omega_{q'_1\mathcal{P}'_2 tr'_1 q'''_2} | q_2 \le q''_2} P_{q'_1\mathcal{P}'_2 tr'_1 q'''_2}[q''_2] = P_{q_2}^{\mathcal{O}_{q'_1\mathcal{P}'_2 tr'_1 q'''_2}}.
\tag{8.53}
$$

Thus, from Equations (8.52) and (8.53), Expression (8.51) can be rewritten into

$$
\begin{aligned}
&\sum_{(q'_1,\mathcal{P}'_2)\in \mathit{Active}(i), N(q'_1)} \; \sum_{tr'_1 \in \Omega^{q'_1}_{H_1}} w_i(q'_1, \mathcal{P}'_2) P^{q'_1}_{H_1}[tr'_1] \\
&\qquad \left(\sum_{q'''_2 \in \Omega'_2 | q_2 \le q'''_2} P'_2[q'''_2] + \sum_{q'''_2 \in \Omega'_2 | q'''_2 < q_2} P'_2[q'''_2] P_{q_2}^{\mathcal{O}_{q'_1\mathcal{P}'_2 tr'_1 q'''_2}} \right) \\
&+ \sum_{(q'_1,\mathcal{P}'_2)\in \mathit{Active}(i), N(q'_1)} \; \sum_{q'''_2 \in \Omega'_2 | q_2 \le q''_2} w_i(q'_1, \mathcal{P}'_2) P^{q'_1}_{H_1}[\delta] P'_2[q'''_2] \\
&+ \sum_{(q'_1\delta,\mathcal{P}'_2)\in \mathit{Active}(i)} \; \sum_{q'''_2 \in \Omega'_2 | q_2 \le q'''_2} w_i(q'_1\delta, \mathcal{P}'_2) P'_2[q'''_2].
\end{aligned}
\tag{8.54}
$$

By regrouping the subexpressions in (8.54), we obtain

$$\begin{aligned}&\sum_{(q_1',\mathcal{P}_2')\in Active(i),N(q_1')}\ \sum_{tr_1'\in\Omega_{H_1}^{q_1'}}\ \sum_{q_2'''\in\Omega_2'|q_2'''<q_2} w_i(q_1',\mathcal{P}_2')P_{H_1}^{q_1'}[tr_1']P_2'[q_2''']P_{q_2}^{\mathcal{O}_{q_1'\mathcal{P}_2'tr_1'q_2'''}} \\ &+ \sum_{(q_1',\mathcal{P}_2')\in Active(i)}\ \sum_{q_2'''\in\Omega_2'|q_2\leq q_2'''} w_i(q_1',\mathcal{P}_2')P_2'[q_2'''].\end{aligned} \tag{8.55}$$

From Equation (8.30), Expression (8.55) can be rewritten into

$$\begin{aligned}&\sum_{(q_1',\mathcal{P}_2')\in Active(i),N(q_1')}\ \sum_{tr_1'\in\Omega_{H_1}^{q_1'}}\ \sum_{q_2'''\in\Omega_2'|q_2'''<q_2} W_{q_1'\mathcal{P}_2'tr_1'q_2'''}(q_2) \\ &+ \sum_{(q_1',\mathcal{P}_2')\in Active(i)}\ \sum_{q_2'''\in\Omega_2'|q_2\leq q_2'''} w_i(q_1',\mathcal{P}_2')P_2'[q_2'''].\end{aligned} \tag{8.56}$$

The induction hypothesis is now sufficient to conclude the validity of (8.47). From an alternative characterization of the set $\{q_2'' \in \Omega_2 \mid q_2'' < q_2\}$ in Expressions (8.47) and (8.45), and by combining (8.45) and (8.47), we obtain

$$\begin{aligned}&\sum_{q_2'\in\Omega_{m(i+1)}|q_2\leq q_2'} P_{m(i+1)}[q_2'] \\ &= \sum_{j\leq i}\ \sum_{(q_1,\mathcal{P}_2)\in Active(j)}\ \sum_{tr_1\in\Omega_{H_1}^{q_1}}\ \sum_{q_2''\in\Omega_2|q_2''\neq q_2,q_2''\in reach(q_1,\mathcal{P}_2,tr_1,q_2)} W_{q_1\mathcal{P}_2tr_1q_2''}(q_2).\end{aligned} \tag{8.57}$$

Observe that the right expression of (8.57) contains a subset of the terms of the right expression of Equation (8.32). This is enough to conclude the validity of (8.35).

2. For each i, $m(i) \leq m(i+1)$.

   This result follows directly from Equation (8.57). In fact, for each state q_2 of H_2, Expression (8.57) for $m(i+1)$ contains a subset of the terms of the Expression (8.57) for $m(i)$.

3. For each state q of H_2, $\lim_{i\to\infty}\sum_{q'\in\Omega_i|q\leq q'} P_i[q'] = P_H[C_q]$.

   This result follows directly from Expression (8.57). In fact, as $i \to \infty$, the right expression of (8.57) converges to the right expression of (8.32).

4. For each i, $m(i) = \sum_{\mathcal{P}\in S(i)} P_{S(i)}[\mathcal{P}]\mathcal{P}$.

   For $i = 0$ the result is trivial. For $i > 0$, from Equation (8.27), $m(i+1)$ is rewritten into.

$$\sum_{(q_1,\mathcal{P}_2)\in Active(i)} w_i(q_1,\mathcal{P}_2)\mathcal{P}_{q_1\mathcal{P}_2}. \tag{8.58}$$

From Equation (8.21), Expression (8.58) can be rewritten into

$$\sum_{(q_1,\mathcal{P}_2)\in Active(i)} w_i(q_1,\mathcal{P}_2)\left(\sum_{tr_1\in\Omega_{H_1}^{q_1}} P_{H_1}^{q_1}[tr_1]\mathcal{P}_{q_1\mathcal{P}_2tr_2} + P_{H_1}^{q_1}[\delta]\delta_{\mathcal{P}_2}\right). \tag{8.59}$$

From Equation (8.17) applied to $\mathcal{P}_{q_1\mathcal{P}_2 tr_2}$ and Equations (8.15) and (8.19) applied to $P_{H_1}^{q_1}[\delta]\delta_{\mathcal{P}_2}$, Expression (8.59) can be rewritten into

$$\sum_{(q_1,\mathcal{P}_2)\in Active(i)} w_i(q_1,\mathcal{P}_2)\left(\sum_{tr_1\in\Omega_{H_1}^{q_1}} P_{H_1}^{q_1}[tr_1]\left(\sum_{\mathcal{P}\in\Omega^S_{q_1\mathcal{P}_2 tr_1}} P^S_{q_1\mathcal{P}_2 tr_1}[\mathcal{P}]\mathcal{P}\right)+\right. \tag{8.60}$$
$$\left. P_{H_1}^{q_1}[\delta]\sum_{\mathcal{P}\in\Omega_{\delta_{\mathcal{P}_2}}} P_{\delta_{\mathcal{P}_2}}[\mathcal{P}]\mathcal{P}\right).$$

From Equation (8.22), Expression (8.60) can be rewritten into

$$\sum_{(q_1,\mathcal{P}_2)\in Active(i)} w_i(q_1,\mathcal{P}_2)\left(\sum_{\mathcal{P}\in\Omega^S_{q_1\mathcal{P}_2}} P^S_{q_1\mathcal{P}_2}[\mathcal{P}]\mathcal{P}\right). \tag{8.61}$$

Finally, from Equation (8.28), Expression (8.61) can be rewritten into

$$\sum_{\mathcal{P}\in\Omega_{S(i+1)}} P_{S(i+1)}[\mathcal{P}]\mathcal{P}, \tag{8.62}$$

which is what we needed to show.

5. For each i, $\mathit{fringe}(H_1,i)\sqsubseteq_{\mathcal{R}} S(i)$ via w_i.

   For $i=0$ the result is trivial. By applying the definitions of a fringe and of $\mathit{fringe}(H_1,i{+}1)$,

$$\begin{aligned}&\mathit{fringe}(H_1,i+1)\\ &\quad=\sum_{q_1\in states(H_2)||q_2|=i\text{or } q_2=q_2'\delta,|q_2|<i} P_{H_1}[C_{q_1}]\mathcal{P}_{q_1}\\ &\quad=\sum_{(q_1,\mathcal{P}_2)\in Active(i)} w_i(q_1,\mathcal{P}_2)\mathcal{P}_{q_1}.\end{aligned}$$

   From (8.28),

$$S(i+1)=\sum_{(q_1,\mathcal{P}_2)\in Active(i)} w_i(q_1,\mathcal{P}_2)\mathcal{P}^S_{q_1\mathcal{P}_2}.$$

   Since for each pair $(q_1,\mathcal{P}_2)$ of $Active(i)$, $\mathcal{P}_{q_1}\sqsubseteq_{\mathcal{R}}\mathcal{P}^S_{q_1\mathcal{P}_2}$ via $w_{q_1,\mathcal{P}_2}$, from Lemma 8.2.1,

$$\sum_{(q_1,\mathcal{P}_2)\in Active(i)} w_i(q_1,\mathcal{P}_2)\mathcal{P}_{q_1}\sqsubseteq_{\mathcal{R}}\sum_{(q_1,\mathcal{P}_2)\in Active(i)} w_i(q_1,\mathcal{P}_2)\mathcal{P}^S_{q_1\mathcal{P}_2}$$

   via $\sum_{(q_1,\mathcal{P}_2)\in Active(i)} w_i(q_1,\mathcal{P}_2)w_{q_1\mathcal{P}_2}$, which is w_{i+1}. ■

6. For each i, each $q_1 \in \mathit{fringe}(H_1, i)$, and each $q_2 \in \mathit{states}(H_2)$, if $W_i(q_1, q_2') = 0$ for each prefix or extension q_2' of q_2, then, for each extension q_1' of q_1 such that $q_1' \in \mathit{fringe}(H_1, i{+}1)$ and each prefix or extension q_2' of q_2, $W_{i+1}(q_1', q_2') = 0$.

   Suppose by contradiction that there is an extension q_1' of q_1 such that $q_1' \in \mathit{fringe}(H_1, i{+}1)$ and a prefix or extension q_2' of q_2 such that $W_{i+1}(q_1', q_2') > 0$. From the definition of W_i and Equation (8.29),

   $$W_{i+1}(q_1, q_2') = \sum_{\mathcal{P}} \sum_{(\bar{q}_1, \mathcal{P}_2) \in \mathit{Active}(i)} w_i(\bar{q}_1, \mathcal{P}_2) w_{\bar{q}_1, \mathcal{P}_2}(q_1, \mathcal{P}) P[q_2']. \qquad (8.63)$$

   Since $W_i(q_1, q_2') > 0$, then there is at least one probability space $\mathcal{P}$ and one pair $(\bar{q}_1, \mathcal{P}_2) \in \mathit{Active}(i)$ such that $w_i(\bar{q}_1, \mathcal{P}_2) > 0$, $w_{\bar{q}_1, \mathcal{P}_2}(q_1, \mathcal{P}) > 0$, and $P[q_2'] > 0$. Then there is at least one prefix q_2'' of q_2' such that $P_2[q_2''] > 0$, which means that $W_i(\bar{q}_1, q_2'') > 0$. However, this is a contradiction since q_2'' is either a prefix or a suffix of q_2.

The execution correspondence theorem can be stated and proved similarly for weak and strong probabilistic simulations. The proofs are simpler than the proof presented in this section, and thus we omit them from this thesis.

8.6.4 Transitivity of Probabilistic Forward Simulations

Now we have enough machinery to prove that probabilistic forward simulations are transitive, i.e., if $M_1 \sqsubseteq_{FS} M_2$ and $M_2 \sqsubseteq_{FS} M_3$, then $M_1 \sqsubseteq_{FS} M_3$. We start by proving a lemma.

Lemma 8.6.2 *Let (H_1, H_2, m, S) be an execution correspondence structure via the probabilistic forward simulation $\mathcal{R}$, and suppose that H_1 represents a weak combined transition $s \overset{a}{\Longrightarrow}_{\mathrm{C}} \mathcal{P}_1$. Then H_2 represents a weak combined transition $s' \overset{a}{\Longrightarrow}_{\mathrm{C}} \mathcal{P}_2$ and there is a probability space $\mathcal{P}_2^S$ such that*

1. *$\mathcal{P}_1 \sqsubseteq_{\mathcal{R}} \mathcal{P}_2^S$ and*
2. *$\mathcal{P}_2 = \sum_{\mathcal{P} \in \Omega_2^S} P_2^S[\mathcal{P}]\mathcal{P}$.*

Proof. Let w_i be the weight functions for $\mathit{fringe}(H_1, i) \sqsubseteq_{\mathcal{R}} S(i)$. Let $\mathcal{P}_1'$ be $\delta\text{-}\mathit{strip}(\mathcal{P}_{H_1})$, $\mathcal{P}_2'$ be $\delta\text{-}\mathit{strip}(\mathcal{P}_{H_2})$, and let

$$\mathcal{P}_{2,S}' \triangleq \sum_{\alpha\delta \in \Omega_{H_1}} \sum_{\mathcal{P} \mid w_{|\alpha|+1}(\alpha\delta, \mathcal{P}) > 0} w_{|\alpha|+1}(\alpha\delta, \mathcal{P})\mathcal{P}. \qquad (8.64)$$

For each $\alpha\delta \in \Omega_{H_1}$ and each $\mathcal{P} \in \mathit{Probs}(\mathit{extstates}(H_2))$, let $w(\alpha\delta, \mathcal{P}) \triangleq w_{|\alpha|+1}(\alpha\delta, \mathcal{P})$.

We show that w is a weight function from $\mathcal{P}_1'$ to $\mathcal{P}_{2,S}'$ and that $\mathcal{P}_{2,S}'$ is well defined. This implies that $\mathcal{P}_1' \sqsubseteq_{\mathcal{R}} \mathcal{P}_{2,S}'$. Then we show that for each element $\alpha\delta$ of Ω_{H_2}, $\sum_{\mathcal{P} \in \Omega_{2,S}'} P_{2,S}'[\mathcal{P}]P[\alpha\delta] = P_{H_2}[C_{\alpha\delta}]$. Since all the elements of the probability spaces of $\Omega_{2,S}'$ end with δ, we obtain that $\mathcal{P}_2'$ is well defined and that $\mathcal{P}_2' = \sum_{\mathcal{P} \in \Omega_{2,S}'} P_{2,S}'[\mathcal{P}]\mathcal{P}$. Then the lemma is proved by defining $\mathcal{P}_1$ to be $\mathit{lstate}(\mathcal{P}_1')$, $\mathcal{P}_2$ to be $\mathit{lstate}(\mathcal{P}_2')$, and $\mathcal{P}_{2,S}$ to be $\mathit{lstate}(\mathcal{P}_{2,S}')$.

To show that w is a weight function we have to verify the three conditions of the definition of a weight function. If $w(\alpha\delta, \mathcal{P}) > 0$, then, from the definition of w, $w_{|\alpha|+1}(\alpha\delta, \mathcal{P}) > 0$.

Since $w_{|\alpha|+1}$ is a weight function, then $\alpha\delta \; \mathcal{R} \; \mathcal{P}$. Let $\mathcal{P} \in \Omega'_{2,S}$. Then $\sum_{\alpha\delta\in\Omega_{H_1}} w(\alpha\delta, \mathcal{P}) = \sum_{\alpha\delta\in\Omega_{H_1}} w_{|\alpha|+1}(\alpha\delta, \mathcal{P})$, which is $P'_{2,S}[\mathcal{P}]$ by definition of $\mathcal{P}'_{2,S}$. Consider now an element $\alpha\delta$ of Ω_{H_1}. Then, $\sum_{\mathcal{P}\in\Omega'_{2,S}} w(\alpha\delta, \mathcal{P}) = \sum_{\mathcal{P}\in\Omega'_{2,S}} w_{|\alpha|+1}(\alpha\delta, \mathcal{P})$. Since $w_{|\alpha|+1}$ is a weight function, then the sum above gives $P_{H_1}[C_{\alpha\delta}] = P'_1[\alpha\delta]$. To show that $\mathcal{P}'_{2,S}$ is well defined we need to show that $\sum_{\alpha\delta\in\Omega_{H_1}} \sum_{\mathcal{P}|w_{|\alpha|+1}(\alpha\delta,\mathcal{P})>0} w_{|\alpha|+1}(\alpha\delta, \mathcal{P}) = 1$. This follows immediately from the fact that w is a weight function and that, since H_1 represents a weak combined transition, $\sum_{\alpha\delta\in\Omega_{H_1}} P'_1[\alpha\delta] = 1$.

We are left to show that for each element $\alpha\delta$ of Ω_{H_2}, $\sum_{\mathcal{P}\in\Omega'_{2,S}} P'_{2,S}[\mathcal{P}]P[\alpha\delta] = P_{H_2}[C_{\alpha\delta}]$. Observe that for each element $\alpha\delta$ of Ω_{H_1}, if $i \leq |\alpha|$ then $w_i(\alpha\delta, \mathcal{P})$ is undefined for each $\mathcal{P}$, and if $i > |\alpha|$, then for each $j \geq i$ and each $\mathcal{P}$, $w_i(\alpha\delta, \mathcal{P})$ is defined iff $w_j(\alpha\delta, \mathcal{P})$ is defined, and if $w_i(\alpha\delta, \mathcal{P})$ is defined then $w_i(\alpha\delta, \mathcal{P}) = w_j(\alpha\delta, \mathcal{P})$. Thus, if we extend each w_i by setting it to 0 whenever it is not defined, then, for each $\alpha\delta \in \Omega_{H_2}$,

$$\sum_{\mathcal{P}\in\Omega'_{2,S}} P'_{2,S}[\mathcal{P}]P[\alpha\delta] = \sum_{\mathcal{P}\in\Omega'_{2,S}} \left(\lim_{i\to\infty} \sum_{\alpha\delta\in\Omega_{H_1}} w_i(\alpha\delta, \mathcal{P}) \right) P[\alpha\delta]. \tag{8.65}$$

Since for each i, w_i is a weight function, and since from the definition of $\mathcal{P}'_{2,S}$ each element $\mathcal{P}$ for which $w_i(\alpha\delta, \mathcal{P}) > 0$ is in $\Omega'_{2,S}$, then we derive

$$\sum_{\mathcal{P}\in\Omega'_{2,S}} P'_{2,S}[\mathcal{P}]P[\alpha\delta] = \sum_{\mathcal{P}\in\Omega'_{2,S}} \left(\lim_{i\to\infty} P_{S(i)}[\mathcal{P}] \right) P[\alpha\delta]. \tag{8.66}$$

By exchanging the limit with the sum and by using Condition 3 of the definition of an execution correspondence structure, the equation above can be rewritten into

$$\sum_{\mathcal{P}\in\Omega'_{2,S}} P'_{2,S}[\mathcal{P}]P[\alpha\delta] = \lim_{i\to\infty} m(i)[\alpha\delta], \tag{8.67}$$

which gives the desired result after using Condition 2 of the definition of an execution correspondence structure. ■

Proposition 8.6.3 *Probabilistic forward simulations are transitive.*

Proof. Let $\mathcal{R}_1$ be a probabilistic forward simulation from M_1 to M_2, and let $\mathcal{R}_2$ be a probabilistic forward simulation from M_2 to M_3. Define $\mathcal{R}$ so that $s_1 \; \mathcal{R} \; \mathcal{P}_3$ iff there is a probability space $\mathcal{P}_2$, and a probability space $\mathcal{P}_3^S$, such that

1. $s_1 \; \mathcal{R}_1 \; \mathcal{P}_2$,
2. $\mathcal{P}_2 \sqsubseteq_{\mathcal{R}_2} \mathcal{P}_3^S$, and
3. $\mathcal{P}_3 = \sum_{\mathcal{P}\in\Omega_3^S} P_3^S[\mathcal{P}]\mathcal{P}$.

We need to show that $\mathcal{R}$ is a probabilistic forward simulation from M_1 to M_3. For this purpose, let $s_1 \; \mathcal{R} \; \mathcal{P}_3$, and let $\mathcal{P}_2$ and $\mathcal{P}_3^S$ satisfy the three conditions above. Let $s_1 \xrightarrow{a} \mathcal{P}_1$. Let M'_2 be obtained from M_2 by introducing a new state s'_2 and by adding a transition $s'_2 \xrightarrow{\tau} \mathcal{P}_2$, where τ is an internal action; similarly, let M'_3 be obtained from M_3 by introducing a new state s'_3 and by adding a transition $s'_3 \xrightarrow{\tau} \mathcal{P}_3$, where τ is an internal action. Let $\mathcal{R}'_1$ be obtained from $\mathcal{R}_1$ by adding the pair $(s_1, \mathcal{D}(s'_2))$, and let $\mathcal{R}'_2$ be obtained from $\mathcal{R}_2$ by adding the pair $(s'_2, \mathcal{D}(s'_3))$. Observe that $\mathcal{R}'_1$ is a probabilistic forward simulation from M_1 to M'_2 and that $\mathcal{R}'_2$ is a probabilistic forward simulation from M'_2 to M'_3.

We want to find two probability spaces $\mathcal{P}'_3$ and $\mathcal{P}'_{3,S}$ such that $s'_3 \overset{a}{\Longrightarrow}_{\mathrm{C}} \mathcal{P}'_3$, $\mathcal{P}'_1 \sqsubseteq_{\mathcal{R}} \mathcal{P}'_{3,S}$, and $\mathcal{P}'_3 = \sum_{\mathcal{P}\in\Omega'_{3,S}} P'_{3,S}[\mathcal{P}]\mathcal{P}$. From the definition of a weak transition, this is sufficient to show that for each state s of $\mathcal{P}_3$ there is a weak combined transition $s \overset{a}{\Longrightarrow}_{\mathrm{C}} \mathcal{P}_s$ of M_3 such that $\mathcal{P}'_3 = \sum_{s\in\Omega_3} P_3[s]\mathcal{P}_s$.

Since $\mathcal{R}'_1$ is a probabilistic forward simulation, there is a weak combined transition $s'_2 \overset{a}{\Longrightarrow}_{\mathrm{C}} \mathcal{P}'_2$ of M'_2 and a probability space $\mathcal{P}'_{2,S}$ such that

$$\mathcal{P}'_2 = \sum_{\mathcal{P}\in\Omega'_{2,S}} P'_{2,S}[\mathcal{P}]\mathcal{P} \quad \text{and} \quad \mathcal{P}'_1 \sqsubseteq_{\mathcal{R}_1} \mathcal{P}'_{2,S}. \tag{8.68}$$

Let H_2 be the probabilistic execution fragment of M'_2 that represents the weak combined transition $s'_2 \overset{a}{\Longrightarrow}_{\mathrm{C}} \mathcal{P}'_2$. Then, by definition of H_2, $\mathcal{P}'_2 = \mathit{lstate}(\delta\text{-}\mathit{strip}(\mathcal{P}_{H_2}))$ (cf. Section 4.2.7).

From the Execution Correspondence Theorem there is an execution correspondence structure (H_2, H_3, m, S), where H_3 is a probabilistic execution fragment of M'_3 that starts from s'_3. From Lemma 8.6.2, H_3 represents a weak combined transition $s'_3 \overset{a}{\Longrightarrow}_{\mathrm{C}} \mathcal{P}''_3$ for same probability space $\mathcal{P}''_3$. Moreover, there is a probability space $\mathcal{P}''_{3,S}$ such that

$$\mathcal{P}''_3 = \sum_{\mathcal{P}\in\Omega''_{3,S}} P''_{3,S}[\mathcal{P}]\mathcal{P} \quad \text{and} \quad \mathcal{P}'_2 \sqsubseteq_{\mathcal{R}_2} \mathcal{P}''_{3,S}. \tag{8.69}$$

Let w_2 be the weight function for $\mathcal{P}'_2 \sqsubseteq_{\mathcal{R}_2} \mathcal{P}''_{3,S}$. For each probability space $\mathcal{P}$ of $\Omega'_{2,S}$, let $w_{\mathcal{P}} : \mathit{states}(M_2) \times \mathit{Probs}(\mathit{states}(M_3)) \to [0,1]$ be a function that is non-zero only in the set $\Omega \times \Omega''_{3,S}$ and such that for each pair $(s, \mathcal{P}')$ of $\Omega \times \Omega''_{3,S}$,

$$w_{\mathcal{P}}(s, \mathcal{P}') = \frac{P[s]w_2(s, \mathcal{P}')}{P'_2[s]}. \tag{8.70}$$

Also, for each probability space $\mathcal{P}$ of $\Omega'_{2,S}$, let

$$\mathcal{P}^{\mathcal{P}}_{3,S} \triangleq \sum_{s\in\Omega} \sum_{\mathcal{P}'\in\Omega''_{3,S}} w_{\mathcal{P}}(s, \mathcal{P}')\mathcal{D}(\mathcal{P}'), \tag{8.71}$$

and let

$$\mathcal{P}^{\mathcal{P}}_3 \triangleq \sum_{\mathcal{P}'\in\Omega^{\mathcal{P}}_{3,S}} P^{\mathcal{P}}_{3,S}[\mathcal{P}']\mathcal{P}'. \tag{8.72}$$

Let $\mathcal{P}'_{3,S}$ be the discrete probability space where $\Omega'_{3,S} = \{\mathcal{P}^{\mathcal{P}}_3 \mid \mathcal{P} \in \Omega_{2,S}\}$, and for each element $\mathcal{P}^{\mathcal{P}}_3$ of $\Omega'_{3,S}$, $P'_{3,S}[\mathcal{P}^{\mathcal{P}}_3] = \sum_{\mathcal{P}'\in\Omega'_{2,S} | \mathcal{P}^{\mathcal{P}}_3 = \mathcal{P}^{\mathcal{P}'}_3} P'_{2,S}[\mathcal{P}']$. Then, the following properties are true.

1. For each probability space $\mathcal{P}$ of $\Omega'_{2,S}$, $w_{\mathcal{P}}$ is a weight function from $\mathcal{P}$ to $\mathcal{P}^{\mathcal{P}}_{3,S}$.

   We verify separately each one of the conditions that a weight function must satisfy.

(a) For each $s \in \mathit{states}(M_2)$, $P[s] = \sum_{\mathcal{P}' \in \mathit{Probs}(\mathit{states}(M_3))} w_{\mathcal{P}}(s, \mathcal{P}')$.

From the definition of $w_{\mathcal{P}}$, the right expression above can be rewritten into

$$\sum_{\mathcal{P}' \in \mathit{Probs}(\mathit{states}(M_3))} \frac{P[s] w_2(s, \mathcal{P}')}{P'_2[s]}. \tag{8.73}$$

Since w_2 is a weight function, $\sum_{\mathcal{P}' \in \mathit{Probs}(\mathit{states}(M_3))} w_2(s, \mathcal{P}') = P'_2[s]$, and thus Expression 8.73 becomes $P[s]$.

(b) For each $\mathcal{P}' \in \mathit{Probs}(\mathit{states}(M_3))$, $\sum_{s \in \mathit{states}(M_2)} w_{\mathcal{P}}(s, \mathcal{P}') = P^{\mathcal{P}}_{3,S}[\mathcal{P}']$.

From Equation (8.71), $P^{\mathcal{P}}_{3,S}[\mathcal{P}'] = \sum_{s \in \Omega} w_{\mathcal{P}}(s, \mathcal{P}')$. Since $w_{\mathcal{P}}$ is non-zero only when the first argument is in Ω, $P^{\mathcal{P}}_{3,S}[\mathcal{P}'] = \sum_{s \in \mathit{states}(M_2)} w_{\mathcal{P}}(s, \mathcal{P}')$.

(c) For each $(s, \mathcal{P}') \in \mathit{states}(M_2) \times \mathit{Probs}(\mathit{states}(M_3))$, if $w_{\mathcal{P}}(s, \mathcal{P}') > 0$ then $s \; \mathcal{R}_2 \; \mathcal{P}'$.

If $w_{\mathcal{P}}(s, \mathcal{P}') > 0$, then, from Equation (8.70), $w_2(s, \mathcal{P}') > 0$. Since w_2 is a weight function, then $s \; \mathcal{R}_2 \; \mathcal{P}'$.

2. $\sum_{\mathcal{P} \in \Omega'_{3,S}} P'_{3,S}[\mathcal{P}]\mathcal{P} = \mathcal{P}''_3$.

From the definition of $\mathcal{P}'_{3,S}$, Equation (8.72), Equation (8.71), and Equation (8.70), $\sum_{\mathcal{P} \in \Omega'_{3,S}} P'_{3,S}[\mathcal{P}]\mathcal{P}$ can be rewritten into

$$\sum_{\mathcal{P} \in \Omega'_{2,S}} \sum_{\mathcal{P}' \in \Omega''_{3,S}} \sum_{s \in \mathit{states}(M_2)} P'_{2,S}[\mathcal{P}] \frac{P[s] w_2(s, \mathcal{P}')}{P'_2[s]} \mathcal{P}'. \tag{8.74}$$

From (8.68), Expression (8.74) can be rewritten into

$$\sum_{\mathcal{P}' \in \Omega''_{3,S}} \sum_{s \in \mathit{states}(M_2)} \frac{P'_2[s] w_2(s, \mathcal{P}')}{P'_2[s]} \mathcal{P}'. \tag{8.75}$$

After simplifying $P'_2[s]$, since w_2 is a weight function from $\mathcal{P}'_2$ to $\mathcal{P}''_{3,S}$, Expression (8.75) can be rewritten into

$$\sum_{\mathcal{P}' \in \Omega''_{3,S}} P''_{3,S}[\mathcal{P}']\mathcal{P}', \tag{8.76}$$

which can be rewritten into $\mathcal{P}''_3$ using Equation (8.69).

3. For each pair $(s'_1, \mathcal{P})$ such that $s'_1 \; \mathcal{R}_1 \; \mathcal{P}$, $s'_1 \; \mathcal{R}_3 \; \mathcal{P}^{\mathcal{P}}_3$.

This follows directly from 1 and (8.72).

Let $\mathcal{P}'_3$ be $\mathcal{P}''_3$, and define a new weight function $w : \mathit{states}(M_1) \times \mathit{Probs}(\mathit{states}(M_3)) \rightarrow [0, 1]$ such that, for each probability space $\mathcal{P}$ of $\Omega'_{2,S}$, $w(s_1, \mathcal{P}^{\mathcal{P}}_3) = w_1(s_1, \mathcal{P})$. Then, it is easy to check that $\mathcal{P}'_1 \sqsubseteq_{\mathcal{R}} \mathcal{P}'_{3,S}$ via w. This fact, together with 2, is sufficient to complete the proof. ■

8.7 Probabilistic Forward Simulations and Trace Distributions

In this section we show that probabilistic forward simulations are sound for the trace distribution precongruence. Specifically, we show that $M_1 \sqsubseteq_{FS} M_2$ implies $M_1 \sqsubseteq_D M_2$. Thus, since $\sqsubseteq_{FS}$ is a precongruence that is contained in $\sqsubseteq_D$, from the definition of $\sqsubseteq_{DC}$ we obtain that $M_1 \sqsubseteq_{FS} M_2$ implies $M_1 \sqsubseteq_{DC} M_2$.

Proposition 8.7.1 *Let $M_1 \sqsubseteq_{FS} M_2$. Then $M_1 \sqsubseteq_D M_2$.*

Proof. Let $\mathcal{R}$ be a probabilistic forward simulation from M_1 to M_2, and let H_1 be a probabilistic execution of M_1 that leads to a trace distribution $\mathcal{D}_1$. From Lemma 8.6.1, there exists a probabilistic execution H_2 of M_2 and two mappings m, S such that (H_1, H_2, m, S) is an execution correspondence structure for $\mathcal{R}$. We show that H_2 leads to a trace distribution $\mathcal{D}_2$ that is equivalent to $\mathcal{D}_1$.

Consider a cone C_β of $\mathcal{D}_1$. The measure of C_β is given by

$$\sum_{q_1 \in states(H_1) | trace(q_1) = \beta, lact(q_1) = lact(\beta)} P_{H_1}[C_{q_1}]. \tag{8.77}$$

The same value can be expressed as

$$\lim_{i \to \infty} \sum_{q_1 \in fringe(H_1, i) | \beta \leq trace(q_1)} P_{H_1}[C_{q_1}]. \tag{8.78}$$

Consider a cone C_β of $\mathcal{D}_2$. The measure of C_β is given by

$$\sum_{q_2 \in states(H_2) | trace(q_2) = \beta, lact(q_2) = lact(\beta)} P_{H_2}[C_{q_2}]. \tag{8.79}$$

The same value can be expressed as

$$\lim_{i \to \infty} \sum_{q_2 \in m(i) | \beta \leq trace(q_2)} P_{m(i)}[C_{q_2}]. \tag{8.80}$$

The reason for the alternative expression is that at the limit each cone of Expression (8.79) is captured completely. Thus, it is sufficient to show that for each finite β and each i,

$$\sum_{q_1 \in fringe(H_1, i) | \beta \leq trace(q_1)} P_{H_1}[C_{q_1}] = \sum_{q_2 \in m(i) | \beta \leq trace(q_2)} P_{m(i)}[q_2]. \tag{8.81}$$

This is shown as follows. Let w_i be the weight function for $m(i) \sqsubseteq_{\mathcal{R}} S(i)$. Then,

$$\sum_{q \in fringe(H_1, i) | \beta \leq trace(q)} P_{H_1}[C_q] = \sum_{q_1 \in fringe(H_1, i) | \beta \leq trace(q_1)} \sum_{\mathcal{P}_2 \in S(i)} w_i(q_1, \mathcal{P}_2). \tag{8.82}$$

Observe that each probability space of $S(i)$ has objects with the same trace, that each state q of *fringe*(H_1, i) is related to some space of $S(i)$, and that each space of $S(i)$ is related to some state q of *fringe*(H_1, i). Thus, from (8.82),

$$\sum_{q \in fringe(H_1, i) | \beta \leq trace(q)} P_{H_1}[C_q] = \sum_{\mathcal{P}_2 \in S(i) | \exists_{q_2 \in \Omega_2} \beta \leq trace(q_2)} \sum_{q_1 \in fringe(H_1, i)} w_i(q_1, \mathcal{P}_2). \tag{8.83}$$

Since w_i is a weight function, we obtain

$$\sum_{q \in fringe(H_1,i) | \beta \leq trace(q)} P_{H_1}[C_q] = \sum_{\mathcal{P}_2 \in S(i) | \exists_{q_2 \in \Omega_2} \beta \leq trace(q_2)} P_{S(i)}[\mathcal{P}_2]. \tag{8.84}$$

Since in a probability space the probability of the whole sample space is 1, we obtain

$$\sum_{q \in fringe(H_1,i) | \beta \leq trace(q)} P_{H_1}[C_q] = \sum_{\mathcal{P}_2 \in S(i) | \exists_{q_2 \in \Omega_2} \beta \leq trace(q_2)} \sum_{q_2 \in \Omega_2} P_{S(i)}[\mathcal{P}_2] P_2[q_2]. \tag{8.85}$$

From an algebraic manipulation based on Condition 3 of an Execution Correspondence Structure, we obtain

$$\sum_{q \in fringe(H_1,i) | \beta \leq trace(q)} P_{H_1}[C_q] = \sum_{q_2 \in m(i) | \beta \leq trace(q_2)} \sum_{\mathcal{P}_2 \in S(i) | q_2 \in \Omega_2} P_{S(i)}[\mathcal{P}_2] P_2[q_2]. \tag{8.86}$$

Finally, from Condition 3 of an Execution Correspondence Structure again, we obtain Equation (8.81). ■

8.8 Discussion

Strong bisimulation was first defined by Larsen and Skou [LS89, LS91] for reactive processes. Successively it was adapted to the alternating model by Hansson [Han94]. In this thesis we have defined the same strong bisimulation as in [Han94]. The formal definition differs from the definition given by Hansson in that we have used the lifting of a relation to probability spaces as defined by Jonsson and Larsen [JL91].

Strong simulation is similar in style to the satisfaction relation for the probabilistic specification systems of Jonsson and Larsen [JL91]. It is from [JL91] that we have borrowed the idea of the lifting of a relation to a probability space.

The probabilistic versions of our simulation relations are justified both by the fact that a scheduler can combine transitions probabilistically, as we have said in this thesis, and by the fact that several properties, namely the ones specified by the logic PCTL of Hansson and Jonsson [Han94], are valid relative to randomized schedulers iff they are valid relative to deterministic schedulers. This fact was first observed by Segala and Lynch [SL94] and can be proved easily using the results about deterministic and randomized schedulers that we proved in Chapter 5.

The weak probabilistic relations were introduced first by Segala and Lynch [SL94]. No simulation relations abstracting from internal computation were defined before. Probabilistic forward simulations are novel in their definition since it is the first time that a state is related to a probability distribution over states.

Chapter 9

Probabilistic Timed Automata

9.1 Adding Time

So far we have extended labeled transition systems to handle probabilistic behavior; however, we have not addressed any real-time issue yet. The main objective of this chapter is to add time to probabilistic automata.

Following an approach that Abadi and Lamport [AL91] call the "old-fashioned recipe", we address real-time issues by augmenting probabilistic automata with some structure that models passage of time. In particular, we adopt the solution of Lynch and Vaandrager [LV95], where a *timed automaton* is an ordinary automaton whose actions include the positive real numbers. The occurrence of a real number d means that time d elapses. In addition, a timed automaton of [LV95] is required to satisfy two *trajectory axioms*: the first axiom says that if time d can elapse and immediately afterwards time d' can elapse, then time $d + d'$ can elapse; the second axiom says that if time d can elapse, then there is a *trajectory* that allows us to associate every real time in the interval $[0, d]$ with a state.

The introduction of real-time in probabilistic automata presents two main problems.

1. Time is a continuous entity, and the time that elapses between the occurrence of two separate actions may depend on a probability distribution that is not discrete. For example, the response time of a system may be distributed exponentially. On the other hand, the probability distributions that we allow in the untimed model are only discrete.

2. In the untimed model the parallel composition operator is defined only for simple probabilistic automata. Since time-passage is modeled by actions of $\Re^+$, in a simple probabilistic timed automaton it is not possible to let time pass according to some probability distribution.

The first problem could be solved by removing the requirement that the probability distribution associated with a transition is discrete. However, in such case we would need to redevelop the whole theory, while if we force each probability distribution to be discrete we can reuse most of the results of the untimed model. For this reason, we choose to work only with discrete probability distributions and we defer to further work the extension of the model to non-discrete probability distributions (cf. Section 13.2.1).

For the second problem the reader may object that it originates from the choice of using a distinct time-passage action for each amount of time that elapses in a transition, and thus we may conclude that the problem would be solved by using a unique action that expresses passage of time [LV93b] rather than a different action for every time; however, the problem has deeper roots.

Example 9.1.1 (Problems with probabilistic passage of time) Suppose that from state s_1 a probabilistic timed automaton M_1 lets time pass for 1 second with probability 1/2 and for 2 seconds with probability 1/2 before performing an action a, and suppose that from state s_2 a probabilistic timed automaton M_2 lets time pass for 0.5 seconds with probability 1/2 and for 1.5 seconds with probability 1/2 before performing action a. What is the probability distribution on the time that elapses from state (s_1, s_2) of $M_1 \| M_2$ before performing a? What can we say about the projections of a probabilistic execution of $M_1 \| M_2$? The reader may note the similarity with the problems encountered in the definition of parallel composition for general probabilistic automata (cf. Section 4.3.3). ■

In order to simplify the handling of trajectories, in this thesis we impose an additional restriction on the time-passage transitions of a probabilistic timed automaton; namely, each transition involving time-passage is required to lead to a Dirac distribution. Probabilistic behavior associated with passage of time is allowed only within a probabilistic execution. Even though this timed model may appear to be restrictive, it is sufficiently powerful to analyze non-trivial timed properties of randomized algorithms (cf. Chapter 10).

In the rest of this chapter we concentrate on the definition of the timed model as an extension of the probabilistic automata of Chapter 4. Most of the concepts are extensions of the definitions of [LV95] to the probabilistic framework; the non-trivial part of the chapter is the definition of a *probabilistic timed execution*, where some measure-theoretical complications arise.

9.2 The Timed Model

In this section we define probabilistic timed automata as an extension of the probabilistic automata of Chapter 4, and we extend the timed executions of [LV95] to our framework. Due to the complications that arise in the definition of a probabilistic timed execution, we define probabilistic timed executions in a separate section.

9.2.1 Probabilistic Timed Automata

A *probabilistic semi-timed automaton* M is a probabilistic automaton whose set of external actions includes $\Re^+$, the set of positive reals, and whose transitions with some action in $\Re^+$ are non-probabilistic, i.e., they lead to a Dirac distribution. Actions from $\Re^+$ are referred to as *time-passage actions*, while non-time-passage actions are referred to as *discrete actions*. We let $d, d', \ldots$ range over $\Re^+$ and more generally, $t, t', \ldots$ range over the set $\Re \cup \{\infty\}$ of real numbers plus infinity. The set of *visible* actions is defined by $vis(M) \triangleq ext(M) \setminus \Re^+$.

A *probabilistic timed automaton* is a probabilistic semi-timed automaton M that satisfies the following two axioms.

A1 If $s \xrightarrow{d} s'$ and $s' \xrightarrow{d'} s''$, then $s \xrightarrow{d+d'} s''$.

For the second axiom, we need an auxiliary definition of a *trajectory*, which describes the state changes that can occur during time-passage. Namely, if I is any left-closed interval of $\Re$ beginning with 0, then an *I-trajectory* is a function $\omega : I \rightarrow states(M)$, such that

$$\omega(t) \xrightarrow{t'-t} \omega(t') \text{ for all } t, t' \in I \text{ with } t < t'.$$

Thus, a trajectory assigns a state to each time t in the interval I in a "consistent" manner. We define $ltime(\omega)$, the "last time" of ω, to be the supremum of I. We define $fstate(\omega)$ to be $\omega(0)$, and if I is right-closed, we also define $lstate(\omega)$ to be $\omega(ltime(\omega))$. A trajectory for a transition $s \xrightarrow{d} s'$ is a $[0, d]$-trajectory such that $fstate(\omega) = s$ and $lstate(\omega) = s'$. Now we can state the second axiom.

A2 Each time-passage transition $s \xrightarrow{d} s'$ has a trajectory.

A probabilistic timed automaton M is *simple* if M is a simple probabilistic automaton.

Axioms **A1** and **A2** express natural properties of time: Axiom **A1** says that if time can elapse in two transitions, then it can also elapse in a single transition; Axiom **A2** says that if time d can elapse, then it is possible to associate states with all times in the interval $[0, d]$ in a consistent way.

Example 9.2.1 (The patient construction) A simple way to add time to a probabilistic automaton is to add arbitrary self-loop timed transitions to each state of a probabilistic automaton. Specifically, given a probabilistic automaton M, we define $patient(M)$ to be the probabilistic timed automaton M' such that

1. $states(M') = states(M)$,
2. $start(M') = start(M)$,
3. $acts(M') = acts(M) \cup \Re^+$,
4. $trans(M') = trans(M) \cup \{(s, d, s) \mid s \in states(M), d \in \Re^+\}$.

Thus, $patient(M)$ is like M except that an arbitrary amount of time can elapse between two discrete transitions. It is immediate to verify that $patient(M)$ satisfies axioms **A1** and **A2**. The patient construction was first defined for ordinary automata in [VL92]. ■

Example 9.2.2 (Simple restrictions on time passage) The patient construction does not specify any limitations to the way time can elapse. Sometimes we may want to specify upper and lower bounds to the time it takes for some transition to take place. Such a limitation can be imposed easily by augmenting the states of a probabilistic automaton with variables that express the time limitations that are imposed. As an easy example consider a probabilistic automaton M with a unique state s and a unique discrete transition (s, a, s). Suppose that we want to add time to M and impose that action a occurs once every at least 1 time unit and at most 2 time units. Then the corresponding probabilistic timed automaton M' can be specified as follows.

1. $states(M') = \{(s, l, h) \mid 0 \leq l \leq 1, 0 \leq l \leq h \leq 2\}$,

2. $start(M') = \{(s, 0, 2)\}$,

3. $acts(M') = \{a\} \cup \Re^+$,

4. $trans(M') = \{((s, 0, h), a, (s, 1, 2)) \mid 0 \leq h \leq 2\} \cup \{((s, l, h), d, (s, l - d, h - d)) \mid d \leq l \leq h\} \cup \{((s, 0, h), d, (s, 0, h - d))d \leq h\}$.

The variables l and h keep track of the time that must or can elapse before performing a. Time passage decreases both the variables unless they are 0. Action a can occur only when $l = 0$ and leads to a state where $l = 1$. This means that at least 1 time unit must elapse before a can be performed again. No time can elapse if $h = 0$. At thet point the only transition that can be performed is the transition labeled with a. Thus, no more than 2 time units can elapse between the occurrence of two actions a. It is immediate to verify that M' satisfies axioms **A1** and **A2**. ■

9.2.2 Timed Executions

Since a probabilistic timed automaton is also a probabilistic automaton, the executions of the untimed model carry over to the timed case. However, an execution associates states with just a countable number of points in time, whereas the trajectory axiom **A2** allows us to associate states with all real times. Also, our intuition about the executions of a timed system is that visible actions occur at points in time, and that time passes "continuously" between these points. In other words, at each point in time a system is in some state. This leads to the definition of a timed execution.

Timed Executions

A *timed execution fragment* α of a probabilistic timed automaton M is a finite or infinite alternating sequence, $\alpha = \omega_0 a_1 \omega_1 a_2 \omega_2 \cdots$, where

1. Each ω_i is a trajectory and each a_i is a discrete action.

2. If α is a finite sequence then it ends with a trajectory.

3. If ω_i is not the last trajectory in α then its domain is a right-closed interval, and there exists a transition $(lstate(\omega_i), \mathcal{P})$ of M such that $(a, fstate(\omega_{i+1})) \in \Omega$.

A timed execution fragment describes all the discrete changes that occur, plus the evolution of the state during time-passage transitions. If α is a timed execution fragment, then we let $ltime(\alpha)$ denote $\sum_i ltime(\omega_i)$. Note that we allow the case where the domain of the final trajectory is of the form $[0, \infty)$; in this case $ltime(\alpha) = \infty$. We define the initial state of α, $fstate(\alpha)$, to be $fstate(\omega_0)$

A *timed execution* is a timed execution fragment whose first state is a start state.

The timed executions and timed execution fragments of a probabilistic timed automaton can be partitioned into *finite*, *admissible*, and *Zeno* timed executions and timed execution fragments. A timed execution (fragment) α is *finite*, if it is a finite sequence and the domain of its final trajectory is right-closed; a timed execution (fragment) α is *admissible* if $ltime(\alpha) = \infty$; a timed execution (fragment) α is *Zeno* if it is neither finite nor admissible.

There are basically two types of Zeno timed executions: those containing infinitely many discrete actions in finite time, and those containing finitely many discrete actions and for which the time interval associated with the last trajectory is right-open. Thus, Zeno timed executions represent executions of a probabilistic timed automaton where an infinite amount of activity occurs in a bounded period of time. (For the second type of Zeno timed executions, the infinitely many time-passage transitions needed to span the right-open interval should be thought of the "infinite amount of activity".)

We will be interested mostly in the admissible timed executions of a probabilistic timed automaton since they correspond to our intuition that time is a force beyond our control that happens to approach infinity. However, according to our definition of a probabilistic timed automaton, it is possible to specify probabilistic timed automata in which from some states no admissible timed execution fragments are possible. This can be because only Zeno timed execution fragments are possible from that state, or because time cannot advance at all (in which case a *time deadlock* has occurred). Although Zeno timed executions are usually non-desirable, research experience has shown that the analysis of a model would be more complicated if Zeno timed executions are ruled out.

Denote by $\textit{t-frag}^*(M)$, $\textit{t-frag}^\infty(M)$, and $\textit{t-frag}(M)$ the sets of finite, admissible, and all timed execution fragments of M. Similarly, denote by $\textit{t-exec}^*(M)$, $\textit{t-exec}^\infty(M)$, and $\textit{t-exec}(M)$ the sets of finite, admissible, and all timed executions of M.

A *timed extended execution fragment* of M, denoted by α, is either a timed execution fragment of M or a sequence $\alpha'\delta$ where α' is a timed execution fragment of M. Denote by $\textit{t-exec}_\delta^*(M)$ and $\textit{t-exec}_\delta(M)$ the sets of finite and all timed extended executions of M.

Concatenations, Prefixes and Suffixes

If ω is an I-trajectory where I is right-closed, and ω' is an I'-trajectory such that $\textit{lstate}(\omega) = \textit{fstate}(\omega')$, then ω and ω' can be concatenated. The concatenation, denoted by $\omega\omega'$ is the least trajectory (the trajectory with the smallest domain) ω'' such that $\omega''(t) = \omega(t)$ for $t \in I$, and $\omega''(t + \textit{ltime}(\omega)) = \omega(t)$ for $t \in I'$. It is easy to show that ω'' is a trajectory.

Likewise, we may combine a countable sequence of "compatible" trajectories into one: if ω_i is an I_i-trajectory, $0 \leq i < \infty$, where all I_i are right-closed, and if $\textit{lstate}(\omega_i) = \textit{fstate}(\omega_{i+1})$ for all i, then the infinite concatenation $\omega_1\omega_2\cdots$ is the least function ω such that for all i and all $t \in I_i$, $\omega(t + \sum_{j<i} \textit{ltime}(\omega_j)) = \omega_i(t)$. It is easy to show that ω is a trajectory.

A finite timed execution fragment $\alpha = \omega_0 a_1 \omega_1 \cdots a_n \omega_n$ of M and a timed (extended) execution fragment $\alpha' = \omega'_n a_{n+1} \omega_{n+1} \cdots$ of M can be *concatenated* if $\textit{lstate}(\alpha) = \textit{fstate}(\alpha')$. In this case the concatenation, written $\alpha \frown \alpha'$, is defined to be $\alpha'' \triangleq \omega_0 a_1 \omega_1 \cdots a_n(\omega_n \omega'_n) a_{n+1} \omega_{n+1} \cdots$. It is easy to see that α is a timed (extended) execution fragment of M.

The notion of prefix for timed execution fragments and timed extended execution fragments is defined as follows. A timed (extended) execution fragment α of M is a *prefix* of a timed (extended) execution fragment α' of M, written $\alpha \leq \alpha'$, if either $\alpha = \alpha'$ or α is finite and there exists a timed (extended) execution fragment α'' of M such that $\alpha' = \alpha \frown \alpha''$. Likewise, α is a *suffix* of α' if there exists a finite timed execution fragment α'' such that $\alpha' = \alpha'' \frown \alpha$. Denote α by $\alpha' \triangleright \alpha''$.

The *length* of a timed execution fragment α expresses the number of discrete actions in α. Thus, even though α is admissible or Zeno (and thus not finite), its length may be finite.

Formally, define the length of $\alpha = \omega_0 a_1 \omega_1 a_2 \omega_2 \cdots$ as

$$|\alpha| \triangleq \begin{cases} n & \text{if } \alpha \text{ is a finite sequence and ends in } \omega_n \\ \infty & \text{if } \alpha \text{ is an infinite sequence.} \end{cases}$$

9.3 Probabilistic Timed Executions

Since a probabilistic timed automaton is also a probabilistic automaton, it is possible to talk about the probabilistic executions of a probabilistic timed automaton. However, as we have pointed out already for ordinary executions, a probabilistic execution does not describe completely the evolution of a probabilistic timed automaton since it does not allow us to associate every real time with the states that are reached at that time. We need a structure that extends probabilistic executions in the same way as a timed execution extends an execution. A timed execution differs from an execution in two aspects:

1. a timed execution has trajectories to express passage of time;

2. a timed execution does not contain any time-passage actions.

In particular, a timed execution hides the time-passage transitions that are scheduled in an execution to let time pass. Given a trajectory ω, there are infinitely many ways to schedule time-passage transitions to move in time $\mathit{ltime}(\omega)$ from $\mathit{fstate}(\omega)$ to $\mathit{lstate}(\omega)$ ($\mathit{lstate}(\omega)$ is meaningful only if the domain of ω is right-closed); the trajectory ω represents all those possible ways. In a similar way, a probabilistic timed execution should not contain any information on the specific time-passage transitions that are scheduled. Thus, a probabilistic timed execution should be a structure where each state records the past history and each transition contains information on the trajectories that are spanned till the occurrence of the next action. However, it may be the case that there is no next action since the next trajectory is right-open. This would not be a problem except for the fact that from a state there can be uncountably many right-open trajectories that leave even though they are generated by scheduling time-passage transitions according to a discrete probability distribution.

Example 9.3.1 (Uncountable branching from countable branching) Consider a probabilistic automaton M that can increase or decrease a variable x of its state at a constant speed, and suppose that every one time unit the speed of x can be complemented nondeterministically. A valid scheduler $\mathcal{A}$ for M is a scheduler that every one time unit chooses the sign of the speed of x according to a uniform binary distribution. As a result, there are uncountably many trajectories leaving from the start state of M if we use $\mathcal{A}$ to resolve the nondeterminism. Thus, if in a probabilistic timed execution we do not allow for a trajectory to be split into pieces, the probabilistic timed execution of M generated by $\mathcal{A}$ would have a non-discrete probability distribution in its transition relation. ■

To express the fact that we allow only discrete probability distributions on a scheduler, we define probabilistic timed executions in two steps. First we define probabilistic time-enriched executions, which contain closed trajectories and time-passage actions (the time-passage transitions that are scheduled are visible); then, we remove the time-passage actions from probabilistic time-enriched executions to yield probabilistic timed executions.

At the end of this section we show that probabilistic executions, probabilistic time-enriched executions, and probabilistic timed executions are strongly related. Specifically, we show that each probabilistic execution is a *sampling* of a probabilistic time-enriched execution where the information contained in the trajectories is lost, and that each probabilistic time-enriched execution is sampled by some probabilistic execution. Furthermore, we show that it is possible to define an equivalence relation directly on probabilistic time-enriched executions that expresses the fact that two probabilistic time-enriched executions denote the same probabilistic timed execution (they just schedule time-passage transitions in a different way).

All the equivalence results that we prove in this section allow us to use the kind of probabilistic execution that is best suited for each problem. In particular, we use probabilistic timed executions for the theorems of Chapter 10, and we use probabilistic time-enriched executions and probabilistic executions for the results of Chapters 11 and 12. Due to the purely technical content of the comparison section (Section 9.3.3), the reader may focus just on the definitions and on the informal explanations (Sections 9.3.1 and 9.3.2) at a first reading. Most of the concepts are simple modifications of concepts defined for probabilistic executions.

9.3.1 Probabilistic Time-Enriched Executions

Time-Enriched Executions

Let M be a probabilistic timed automaton. A *time-enriched execution fragment* of M is a finite or infinite alternating sequence $\alpha = \omega_0 a_1 \omega_1 a_2 \omega_2 \cdots$ where

1. The domain of ω_0 is $[0,0]$.

2. Each ω_i is a trajectory with a closed domain and each a_i is an action.

3. If a_i is a visible action, then the domain of ω_i is $[0,0]$, and there exists a transition $(\mathit{lstate}(\omega_{i-1}), \mathcal{P})$ of M such that $(a_i, \mathit{fstate}(\omega_i)) \in \Omega$.

4. If a_i is a time-passage action, then the domain of ω_i is $[0, a_i]$ and $\mathit{lstate}(\omega_{i-1}) = \mathit{fstate}(\omega_i)$.

Denote by $\mathit{te\text{-}frag}^*(M)$ and $\mathit{te\text{-}frag}(M)$ the set of finite and all time-enriched execution fragments of M, respectively. The notation for $\mathit{fstate}(\alpha)$, $\mathit{lstate}(\alpha)$ and $\mathit{ltime}(\alpha)$ extends trivially.

A time-enriched execution fragment α contains more information than a timed execution fragment since it is possible to observe what time-passage transitions are used to generate α.

A time-enriched *extended* execution fragment of M is either a time-enriched execution fragment of M or a sequence $\alpha\delta$ where α is a finite time-enriched execution fragment of M. The notation for $\mathit{lstate}(\alpha)$ extends trivially.

A finite time-enriched execution fragment $\alpha = \omega_0 a_1 \omega_1 \cdots a_n \omega_n$ of M and a time-enriched extended execution fragment $\alpha' = \omega'_n a_{n+1} \omega_{n+1} \cdots$ of M can be *concatenated* if $\mathit{lstate}(\alpha) = \mathit{fstate}(\alpha')$. In this case the concatenation is defined to be $\alpha'' \triangleq \omega_0 a_1 \omega_1 \cdots a_n \omega_n a_{n+1} \omega_{n+1} \cdots$, and is denoted by $\alpha \frown \alpha'$. It is easy to see that α'' is a time-enriched extended execution fragment of M. A time-enriched extended execution fragment α of M is a *prefix* of a time-enriched extended execution fragment α' of M, written $\alpha \leq \alpha'$, if either $\alpha = \alpha'$ or α is finite and there exists a time-enriched extended execution fragment α'' of M such that $\alpha' = \alpha \frown \alpha''$. Likewise, α is a *suffix* of α' if there exists a finite time-enriched execution fragment α'' such that $\alpha' = \alpha'' \frown \alpha$. Denote α by $\alpha' \triangleright \alpha''$.

Time-Enriched Transitions

Let $(s, \mathcal{P})$ be a combined transition of M. For each pair (a, s') of Ω, if a is a discrete action, then let $\mathcal{P}_{(a,s')}$ be $\mathcal{D}((a, s'))$; if a is a time-passage action, then let $\mathcal{P}_{(a,s')}$ be a discrete probability distribution of $\mathit{Probs}(\mathit{trajectories}(M, s, a, s'))$, where $\mathit{trajectories}(M, s, a, s')$ denotes the set of trajectories for $s \xrightarrow{a} s'$. The pair $\sum_{(a,s') \in \Omega} P[(a, s')](s, \mathcal{P}_{(a,s')})$ is called a *time-enriched transition* of M.

Thus, a time-enriched transition adds information to a combined transition by specifying what state is reached at each intermediate time. A combined transition gives just the extremes of a trajectory, dropping all the information about what happens in the middle.

Probabilistic Time-Enriched Executions

A *probabilistic time-enriched execution fragment* H of a timed probabilistic automaton M is a fully probabilistic automaton such that

1. $\mathit{states}(H) \subseteq \mathit{te\text{-}frag}^*(M)$
2. for each transition $tr = (q, \mathcal{P})$ of H there is a time-enriched transition $tr' = (\mathit{lstate}(q), \mathcal{P}')$ of M, called the *corresponding time-enriched transition*, such that $\mathcal{P} = q \frown \mathcal{P}'$.
3. each state of H is reachable and enables one transition.

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

As for the untimed case, there is a strong relationship between the time-enriched extended execution fragments of a probabilistic timed automaton and the extended executions of one of its probabilistic time-enriched execution fragments. Specifically, let M be a probabilistic timed automaton and let H be a probabilistic time-enriched 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

$$\alpha\!\downarrow \;\triangleq\; \begin{cases} q_0 \frown \mathit{lstate}(q_0) a_1 \mathit{ltraj}(q_1) a_2 \cdots & \text{if } \alpha \text{ does not end in } \delta, \\ q_0 \frown \mathit{lstate}(q_0) a_1 \mathit{ltraj}(q_1) a_2 \cdots a_n \mathit{ltraj}(q_n) \delta & \text{if } \alpha = q_0 a_1 q_1 \cdots a_n q_n \delta, \end{cases} \tag{9.1}$$

where $\mathit{ltraj}(q_i)$ denotes the last trajectory of q_i. It is immediate to observe that $\alpha\!\downarrow$ is a time-enriched extended execution fragment of M. For each time-enriched extended execution fragment α of M such that $q_0 \leq \alpha$, i.e., $\alpha = q_0 \frown \omega_0 a_1 \omega_1 \cdots$, let

$$\alpha\!\uparrow\! q_0 \;\triangleq\; \begin{cases} q_0 a_1 (q_0 a_1 \omega_1) a_2 (q_0 a_1 \omega_1 a_2 \omega_2) \cdots & \text{if } \alpha \text{ does not end in } \delta, \\ q_0 a_1 (q_0 a_1 \omega_1) \cdots (q_0 a_1 \omega_1 \cdots a_n \omega_n) \delta & \text{if } \alpha = q_0 a_1 \omega_1 \cdots a_n \omega_n \delta. \end{cases} \tag{9.2}$$

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

Proposition 9.3.1 *Let H be a probabilistic time-enriched execution fragment of a probabilistic timed automaton M. Then, for each extended execution α of H,*

$$(\alpha\downarrow)\uparrow q_0 = \alpha, \tag{9.3}$$

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

$$(\alpha\uparrow q_0)\downarrow = \alpha. \tag{9.4}$$

Events

The probability space $\mathcal{P}_H$ associated with a probabilistic time-enriched execution H is defined as for the untimed case. Thus, Ω'_H is the set of time-enriched extended execution fragments of M that correspond to complete extended executions of H, i.e.,

$$\Omega'_H \triangleq \{\alpha\downarrow \mid \alpha \text{ is a complete extended execution of } H\}, \tag{9.5}$$

where an extended execution α of H is complete iff either α is infinite, or $\alpha = \alpha'\delta$, α' is a finite execution of H, and $\delta \in \Omega^H_{lstate(\alpha)}$. For each finite time-enriched extended execution fragment α of M, let C^H_α denote the *cone*

$$C^H_\alpha \triangleq \{\alpha' \in \Omega_H \mid \alpha \leq \alpha'\}. \tag{9.6}$$

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.,

$$\mathcal{F}'_H \triangleq \sigma(\mathcal{C}_H). \tag{9.7}$$

Define a measure μ on $\mathcal{C}_H$ such that the measure $\mu_H(C^H_\alpha)$ of a cone C^H_α 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

$$\mu_H(C^H_\alpha) \triangleq 1; \tag{9.8}$$

if $\alpha = q_0 \frown \omega_0 a_1 \omega_1 \cdots \omega_{n-1} a_n \omega_n$, then

$$\mu_H(C^H_\alpha) \triangleq P^H_{q_0}[(a_1, q_1)] \cdots P^H_{q_{n-1}}[(a_n, q_n)], \tag{9.9}$$

where for each i, $1 \leq i < n$, $q_i = q_0 \frown \omega_0 a_1 \omega_1 \cdots \omega_{i-1} a_i \omega_i$; if $\alpha = q_0 \frown \omega_0 a_1 \omega_1 \cdots \omega_{n-1} a_n \omega_n \delta$, then

$$\mu_H(C^H_\alpha) \triangleq P^H_{q_0}[(a_1, q_1)] \cdots P^H_{q_{n-1}}[(a_n, q_n)] P_{q_n}[\delta], \tag{9.10}$$

where for each i, $1 \leq i \leq n$, $q_i = q_0 \frown \omega_0 a_1 \omega_1 \cdots \omega_{i-1} a_i \omega_i$. Then the probability measure P'_H is the unique measure on $\mathcal{F}_H$ that extends μ_H, and $\mathcal{P}_H$ is the completion of $\mathcal{P}_H$.

Finite Probabilistic Time-Enriched Executions, Prefixes, Conditionals, and Suffixes

Since a probabilistic time-enriched execution is a fully probabilistic automaton, the definitions of finiteness, prefix, conditional and suffix of Section 4.2.6 extend directly: we just need to define the length of a time-enriched execution fragment α as the number of actions that occur in α.

9.3.2 Probabilistic Timed Executions

We now define the probabilistic timed executions of a probabilistic timed automaton. We use probabilistic time-enriched executions to characterize those transitions that originate from discrete schedulers.

Timed Transitions

A timed transition expresses the result of choosing either an infinite trajectory or a finite trajectory followed by some discrete action at random. However, a timed transition should be the result of scheduling a collection of time-enriched transitions, so that we are guaranteed that it is due to a discrete scheduler. For this reason, we derive a timed transition from the probability distribution associated with a time-enriched probabilistic execution. The derivation proceeds in two steps: first all the time-passage actions are removed and the corresponding trajectories are concatenated; then the resulting structure is truncated at the occurrence of the first action.

Removing Time-Passage Actions. Let $\alpha = \omega_0 a_1 \omega_1 a_2 \omega_2 \cdots$ be a time-enriched execution fragment of a probabilistic timed automaton M. The timed execution represented by α, denoted by $t\text{-}exec(\alpha)$, is the sequence obtained from α by removing all the time-passage actions and by concatenating all the trajectories whose intermediate action is removed.

Let H be a probabilistic time-enriched execution fragment of a probabilistic timed automaton M. Let

$$\Omega \stackrel{\triangle}{=} t\text{-}exec(\Omega_H) \cup limits(t\text{-}exec(\Omega_H)), \tag{9.11}$$

where $limits(t\text{-}exec(\Omega_H))$ is the set of timed executions α of M that end with an open trajectory and such that for each finite prefix α' of α there is an element α'' of $t\text{-}exec(\Omega_H)$ such that $\alpha' \leq \alpha''$. Then, $t\text{-}exec(\mathcal{P}_H)$ denotes the probability space $completion((\Omega, \mathcal{F}, P))$ where $\mathcal{F}$ is the σ-field generated by the cones on Ω, and P is $t\text{-}exec(P_H)$.

The reason for the definition of the sample space of $t\text{-}exec(P_H)$ is mainly technical: we want to establish a relationship between probabilistic time-enriched executions and probabilistic timed executions, and we want the relationship to be preserved by projection of probabilistic timed executions in a parallel composition context. Informally, we are interested in a distribution over trajectories, possibly followed by an action, without keeping any information on how such a distribution is obtained. The elements of the sample space that end with right open trajectories can be affected by the way the transitions are scheduled in a probabilistic time-enriched execution. Moreover, these elements of Ω can create problems for parallel composition. Closing the sample space under limit makes such differences invisible. The reader interested in more details is referred to Sections 9.3.3 and 9.5, and specifically to Examples 9.3.3 and 9.5.1.

Example 9.3.2 (What *t-exec* identifies) Figure 9-1 gives an example of two probabilistic time-enriched executions that are mapped to the same structure by $t\text{-}exec()$. We assume to have two functions ω and ω' defined on the real numbers, and we denote by $\omega_{d,d'}$ the trajectory ω'' with domain $[0, d' - d]$ such that for each $t \leq d' - d$, $\omega''(t) = \omega(t - d)$. A similar notation is used for ω'. ■

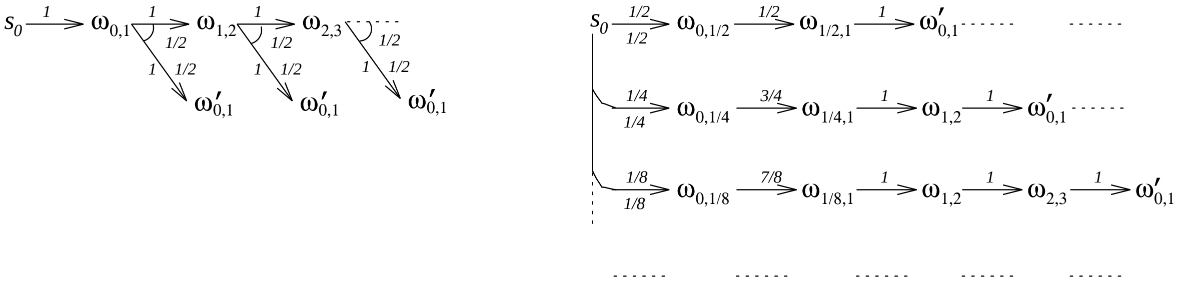


Figure 9-1: Probabilistic time-enriched executions that are mapped to the same structure.

Truncation at the First Action. Let M be a probabilistic timed automaton, and let q be a finite timed execution fragment of M. For each extended timed execution fragment α of M such that $q \leq \alpha$, let

$$\mathit{truncate}_q(\alpha) \triangleq \begin{cases} \alpha & \text{if no action occurs in } \alpha \triangleright q \\ q \frown \omega_0 a_1 \mathit{fstate}(\omega_1) & \text{if } \alpha \triangleright q = \omega_0 a_1 \omega_1 \cdots \end{cases} \tag{9.12}$$

Let H be a probabilistic time-enriched execution fragment of M, and let q be a prefix of the start state of H. Then define $\mathit{truncate}_q(\mathit{t\text{-}exec}(\mathcal{P}_H))$ to be the probability space $\mathcal{P}$ where $\Omega = \mathit{truncate}_q(\mathit{t\text{-}exec}(\Omega_H))$, $\mathcal{F}$ is the σ-field generated by the cones of Ω, and P is the measure $\mathit{truncate}_q(\mathit{t\text{-}exec}(P_H))$.

Timed Transitions. A *timed transition* of M leaving from a state s is a pair $(s, \mathcal{P})$ such that there is a probabilistic time-enriched execution fragment H of M starting in s, and $\mathcal{P} = \mathit{truncate}_s(\mathit{t\text{-}exec}(\mathcal{P}_H))$.

Probabilistic Timed Executions

A *probabilistic timed execution fragment* of a probabilistic timed automaton M, denoted by H, consists of four components.

1. A set $\mathit{states}(H) \subseteq \mathit{t\text{-}frag}_\delta(M)$ of states.
2. A unique start state q_0^H.
3. An action signature $\mathit{sig}(H) = \mathit{sig}(M)$.
4. A transition relation $\mathit{trans}(M)$ consisting of pairs $(q, \mathcal{P})$ such that there exists a timed transition $(\mathit{lstate}(q), \mathcal{P}')$ of M satisfying $\mathcal{P} = q \frown \mathcal{P}'$. Observe that, from the discussion in Section 3.1.5, $q \frown \mathcal{P}'$ is well defined.

Moreover, each state of H is reachable, enables at most one transition, and enables one transition iff it is a finite timed execution fragment of M. A *probabilistic timed execution* of M is a probabilistic timed execution fragment of M whose start state is a start state of M.

An execution of H is a sequence of states of H, $\alpha = q_0 q_1 \cdots$, such that for each i, $q_{i+1} \in \Omega^H_{q_i}$. As for the untimed case, there is a strong correspondence between the timed extended execution fragments of a probabilistic timed execution H of M and the executions of H. Specifically, let

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

$$\alpha\!\downarrow \;\triangleq\; \lim_i q_i, \tag{9.13}$$

where the limit is taken under prefix ordering. It is immediate to observe that $\alpha\!\downarrow$ is a timed extended execution fragment of M. For each timed extended execution fragment α of M such that $q_0 \leq \alpha$, i.e., $\alpha = q_0 \frown \omega_0 a_1 \omega_1 \cdots$, let q_i be $q_0 \frown \omega_0 a_1 \omega_1 \cdots a_i \mathit{fstate}(\omega_i)$, and if $\alpha \triangleright q_0$ is a finite sequence with n discrete actions, let q_{n+1} be α. Then let

$$\alpha\!\uparrow\! q_0 \;\triangleq\; q_0 q_1 q_2 \cdots. \tag{9.14}$$

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

Proposition 9.3.2 *Let H be a probabilistic timed execution fragment of a probabilistic timed automaton M. Then, for each execution α of H,*

$$(\alpha\!\downarrow)\!\uparrow\! q_0 = \alpha, \tag{9.15}$$

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

$$(\alpha\!\uparrow\! q_0)\!\downarrow = \alpha. \tag{9.16}$$

Events

The probability space $\mathcal{P}_H$ associated with a probabilistic timed execution fragment H is defined similarly to the untimed case. The set Ω'_H the set of extended timed execution fragments of M that correspond to complete executions of H, where an execution of H is complete iff it is either infinite or it leads to a state that does not enable any transition. The σ-field $\mathcal{F}'_H$ is the minimum σ-field that contains the class of cones of Ω'_H. The measure P'_H is the unique measure that extends the measure defined on cones as follows: if $\alpha = q_0^H \frown \omega_0 a_1 \omega_1 a_2 \cdots a_n \omega_n$, then

$$P'_H[C_\alpha] = P^H_{q_0}[q_1] \cdots P^H_{q_{n-1}}[q_n] P^H_{q_n}[C_\alpha] \tag{9.17}$$

where for each $i \leq n$, $q_i = q_0^H \frown \omega_0 a_1 \omega_1 \cdots a_n \mathit{fstate}(\omega_i)$; if $\alpha = q_0^H \frown \omega_0 a_1 \omega_1 a_2 \cdots a_n \omega_n \delta$, then

$$P'_H[C_\alpha] = P^H_{q_0}[q_1] \cdots P^H_{q_{n-1}}[q_n] P^H_{q_n}[\alpha] \tag{9.18}$$

where for each $i \leq n$, $q_i = q_0^H \frown \omega_0 a_1 \omega_1 \cdots a_n \mathit{fstate}(\omega_i)$. Observe that although there are uncountably many cones in $\mathcal{F}'_H$, every union of cones is expressible as a countable union of disjoint cones. Then, $\mathcal{P}_H$ is the completion of $\mathcal{P}'_H$.

Finite Probabilistic Timed Executions, Prefixes, Conditionals, and Suffixes

Finiteness and prefix are defined similarly to the untimed case, and thus we do not repeat the definitions here.

Conditionals and suffixes differ in a small detail concerning the start state. The reader should observe the similarity of these definitions to those for the untimed case. Also, observe that the properties of conditionals and suffixes (Propositions 9.3.3 and 9.3.4) are the same as for the untimed case. This is what allows us to extend the results for the untimed case directly to the timed case.

Let H be a probabilistic timed execution fragment of a probabilistic timed automaton M, and let q be a prefix of some state of H such that q_0^H is a prefix of q. Then $H|q$ is a new probabilistic execution fragment defined as follows:

1. $states(H|q) = \{q\} \cup \{q' \in states(H) \mid q \leq q'\}$;
2. $start(H|q) = \{q\}$.
3. for each state q' of $H|q$ different from q, $tr_{q'}^{H|q} = tr_{q'}^{H}$.
4. let $\bar{q}$ be the maximum state of H that is a prefix of q. Then, $tr_q^{H|q} = (q, \mathcal{P}_{\bar{q}}^H | C_q)$.

$H|q$ is called a *conditional* probabilistic timed execution fragment. We show later that $H|q$ is a probabilistic timed execution. Observe 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): the sample spaces are the same, the generators are the same, and the probability measures coincide on the generators. Thus, the following proposition is true.

Proposition 9.3.3 *Let H be a probabilistic timed execution fragment of a probabilistic timed automaton M, and let q be a prefix of a state of H such that $q_0^H \leq q$. 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 $P_H[E] = P_H[C_q]P_{H|q}[E]$.* ■

Let H be a probabilistic timed execution fragment of a probabilistic timed automaton M, and let q be a prefix of some state of H such that q_0^H is a prefix of q. Then $H \rhd q$ is a new probabilistic execution fragment defined as follows:

1. $states(H \rhd q) = \{q' \rhd q \mid q' \in states(H|q)\}$;
2. $start(H|q) = \{lstate(q)\}$.
3. for each state q' of $H \rhd q$, $tr_{q'}^{H \rhd q} = tr_{q ^\frown q'}^{H|q} \rhd q$.

$H \rhd q$ is called a *suffix* of H. It is easy to check that the probability spaces $\mathcal{P}_{H \rhd q}$ and $\mathcal{P}_{H|q}$ are in a one-to-one correspondence through the measurable function $f : \Omega_{H \rhd q} \to \Omega_{H|q}$ such that for each $\alpha \in \Omega_{H \rhd q}$, $f(\alpha) = q ^\frown \alpha$. The inverse of f is also measurable and associates $\alpha \rhd q$ with each timed execution α of $\Omega_{H|q}$. Thus, directly from Proposition 9.3.3, we get the following proposition.

Proposition 9.3.4 *Let H be a probabilistic timed execution fragment of a probabilistic timed automaton M, and let q be a prefix of a state of H such that $q_0^H \leq q$. Then, for each subset E of $\Omega_{H \rhd q}$,*

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

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

We are left with showing that $H|q$ is well defined. The proof of this apparently obvious fact is not simple and contains several technical details.

Proposition 9.3.5 *Let H be a probabilistic timed execution fragment of a probabilistic timed automaton M, and let q be a prefix of a state of H such that $q_0^H \leq q$. Then, $H|q$ is a probabilistic timed execution fragment of M.*

Proof. We just need to verify that the transition leaving from state q in $H|q$ is a timed transition. Let $\bar{q}$ be the maximum state of H that is a prefix of q. Then, from the definition of a timed transition, there is a probabilistic time-enriched execution fragment $H_{\bar{q}}$ of M such that $\mathcal{P}_{\bar{q}}^H = \bar{q} \frown \mathit{truncate}_{\mathit{lstate}(\bar{q})}(\mathit{t\text{-}exec}(\mathcal{P}_{H_{\bar{q}}}))$. From the definition of $\mathit{tr}_q^{H|q}$, we need to find a probabilistic time-enriched execution fragment H_q of M such that

$$(\bar{q} \frown \mathit{truncate}_{\mathit{lstate}(\bar{q})}(\mathit{t\text{-}exec}(\mathcal{P}_{H_{\bar{q}}})))|C_q = q \frown \mathit{truncate}_{\mathit{lstate}(q)}(\mathit{t\text{-}exec}(\mathcal{P}_{H_q})). \qquad (9.19)$$

Let q' be $q \triangleright \bar{q}$. From the definition of $\bar{q}$, q' is just one closed trajectory. Thus, if we build H_q such that

$$(\mathit{t\text{-}exec}(\mathcal{P}_{H_{\bar{q}}}))|C_{q'} = q' \frown \mathit{t\text{-}exec}(\mathcal{P}_{H_q}), \qquad (9.20)$$

then Equation 9.19 follows easily using simple properties of *truncate*. Thus, the rest of this proof is dedicated to the construction of an H_q that satisfies (9.20).

Let $q_1, q_2, \ldots$ be an enumeration of the minimal states q'' of H such that $q' \leq \mathit{t\text{-}exec}(q'')$. We distinguish two cases.

1. For each i, $\mathit{t\text{-}exec}(q_i) = q'$.

   The construction for H_q in this case is carried out in the proof of Proposition 9.3.8 (cf. Equation 9.29). We give a forward pointer to avoid too many technical details at this point.

2. There is an i such that $q' < \mathit{t\text{-}exec}(q_i)$.

   We prove this case by reducing the problem to the previous case. That is, we build a new probabilistic time-enriched execution fragment $H'_{\bar{q}}$ such that $\mathit{t\text{-}exec}(\mathcal{P}_{H_{\bar{q}}}) = \mathit{t\text{-}exec}(\mathcal{P}_{H'_{\bar{q}}})$ and such that the minimal states q'' of $H'_{\bar{q}}$ such that $q' \leq \mathit{t\text{-}exec}(q'')$ satisfy $q' = \mathit{t\text{-}exec}(q')$.

   Recall first that q' is a trajectory whose domain is $[0, d]$ for some $d > 0$. Define a collection of finite time-enriched execution fragments $q'_1, q'_2, \cdots$ as follows: for each i, if $\mathit{t\text{-}exec}(q_i) = q'$ then $q'_i = q_i$; otherwise, represent q_i as $\bar{q}_i \frown \mathit{lstate}(\bar{q}_i) d_i \omega_i$, where $\bar{q}_i$ is a state of $H_{\bar{q}}$, and let q'_i be $\bar{q}_i \frown \mathit{lstate}(\bar{q}_i) d_{i,1} \omega_{i,1} d_{i,2} \omega_{i,2} d_{i,3} \omega_{i,3}$ where $\omega_i = \omega_{i,1} \omega_{i,2} \omega_{i,3}$, $\mathit{t\text{-}exec}(\bar{q}_i \frown \mathit{lstate}(\bar{q}_i) d_{i,1} \omega_{i,1} d_{i,2} \omega_{i,2}) = q'$, and the actions $d_{i,1}$ and $d_{i,2}$ are chosen in such a way that for each i $\bar{q}_i \frown \mathit{lstate}(\bar{q}_i) d_{i,1} \omega_{i,1}$ is not a prefix of any of the q'_j's, $j \neq i$. In other words, we split all the q_i's in such a way that a state that corresponds to q' is reached always and such that none of the states of $H_{\bar{q}}$ are identified. Then,

$$\begin{aligned} \mathit{states}(H'_{\bar{q}}) \;\; = \;\; & \{q'' \mid \exists_i q'' \leq q'_i\} \qquad (9.21) \\ & \cup \left(\bigcup_i \{q'_i \frown (q'' \triangleright q_i) \mid q'' \in \mathit{states}(H_{\bar{q}}), q_i < q''\} \right). \end{aligned}$$

The transition relation of $H'_{\bar{q}}$ is obtained from the transition relation of $H_{\bar{q}}$ by scheduling the same time-enriched transitions of M as before except for the states $\bar{q}_i$ where the intermediate transitions leading to the q'_i's are scheduled. It is simple to check that $H'_{\bar{q}}$ satisfies the desired properties. ■

9.3.3 Probabilistic Executions versus Probabilistic Timed Executions

In this section we show the relationship between probabilistic executions, probabilistic time-enriched executions, and probabilistic timed executions. The main idea is that they all represent the same structures with different levels of detail. We show that a probabilistic execution is a sampling of a probabilistic time-enriched execution, where the information given by the trajectories is lost. Conversely, we show that each probabilistic time-enriched execution is sampled by some probabilistic execution. We show that each probabilistic time-enriched execution represents a probabilistic timed execution and that each probabilistic timed execution is represented by some probabilistic time-enriched execution. Essentially, a probabilistic time-enriched execution is a probabilistic timed execution with the additional information of what time-passage transitions are scheduled. Finally, we define an equivalence relation on probabilistic time-enriched executions that captures the idea of representing the same probabilistic timed execution. This equivalence relation will be useful for parallel composition.

Probabilistic Executions versus Probabilistic Time-Enriched Executions

There is a close relationship between the probabilistic executions of a probabilistic timed automaton and its probabilistic time-enriched executions. Informally, a probabilistic time-enriched execution contains more information than a probabilistic execution because it associates a state with every real time rather than with a countable set of times. In other words, a probabilistic execution can be seen as a sampling of a probabilistic time-enriched execution at countably many points. In later chapters we will see that probabilistic executions are sufficient for the study of the properties of a system whenever such properties do not depend on the actual states that are reached at each time. For the moment we just define what it means for a probabilistic execution to sample a probabilistic time-enriched execution, and we show that each probabilistic time-enriched execution is sampled by some probabilistic execution and that each probabilistic execution samples some probabilistic time-enriched execution. We start by defining a function *sample* that applied to a probabilistic time-enriched execution H of a probabilistic timed automaton M gives a probabilistic execution H' of M, which by definition samples H.

Let $\alpha = \omega_0 a_1 \omega_1 a_2 \omega_2 \cdots$ be a time-enriched execution of a probabilistic timed automaton M, and let $sample(\alpha)$ be the sequence $\alpha' = lstate(\omega_0) a_1 lstate(\omega_1) a_2 lstate(\omega_2) \cdots$. Then, it is easy to check that α' is an execution of M. We say that α' *samples* α. Define

$$states(H') \triangleq sample(states(H)). \tag{9.22}$$

Let $(q, \mathcal{P})$ be a transition of H. Define *sample* on Ω as follows: $sample((a, q')) = (a, sample(q'))$, and $sample(\delta) = \delta$. Then, define the transition $sample((q, \mathcal{P}))$ to be

$$sample((q, \mathcal{P})) \triangleq (sample(q), sample(\mathcal{P})). \tag{9.23}$$

For each state q of H', let $sample^{-1}(q)$ be the set of states q' of H such that $sample(q') = q$. Observe that all the states of $sample^{-1}(q)$ are incomparable under prefix. For each $q' \in sample^{-1}(q)$, let

$$\bar{p}_{q'}^{sample^{-1}(q)} \triangleq \frac{P_H[C_{q'}]}{\sum_{q'' \in sample^{-1}(q)} P_H[C_{q''}]}. \tag{9.24}$$

Then, the transition enabled from q in H' is defined to be

$$tr_q^{H'} \triangleq \sum_{q' \in sample^{-1}(q)} \bar{p}_{q'}^{sample^{-1}(q)} sample(tr_{q'}^{H}). \tag{9.25}$$

Observe the similarity of Equations (9.24) and (9.25) with the equations that the fine the projection of a probabilistic execution (cf. Equations (4.21) and (4.22)).

Proposition 9.3.6 below shows that H' is a probabilistic execution of M. We say that H' *samples* H. Then, Proposition 9.3.7 shows that each probabilistic execution samples some probabilistic time-enriched execution.

Proposition 9.3.6 *For each probabilistic time-enriched execution H of a probabilistic timed automaton M, $sample(H)$ is a probabilistic execution of M.*

Proof. Let H' denote $sample(H)$. The fact that each state of H' is reachable can be shown by a simple inductive argument; the fact that each state of H' is a finite execution fragment of M follows from a simple analysis of the definition of *sample* and of a time-enriched execution.

We need to check that for each state q of H' the transition enabled from q in H' is generated by a combined transition of M. From (9.25), it is enough to show that for each state q' of $sample^{-1}(q)$ the transition $sample(tr_{q'}^H)$ is generated by a combined transition of M.

Since H is a probabilistic time-enriched execution of M, then there is a time-enriched transition $(lstate(q'), \mathcal{P})$ of M such that $\mathcal{P}_{q'}^H = q' \frown \mathcal{P}$. From the definition of *sample* and the definition of a time-enriched transition, $(lstate(q), sample(\mathcal{P}))$ is a combined transition of M, and $sample(\mathcal{P}_{q'}^H) = sample(q') \frown sample(\mathcal{P})$, which means that $sample(\mathcal{P}_q^{H'}) = q \frown sample(\mathcal{P})$. This is enough to conclude. ■

Proposition 9.3.7 *Let H be a probabilistic execution of a probabilistic timed automaton M. Then there is a probabilistic time-enriched execution H' of M such that $H = sample(H')$.*

Proof. We build H' inductively in such a way that for each state q of H there is exactly one state q' of H' in $sample^{-1}(q)$. The start state of H' is the same as the start state of H.

Suppose that the transition relation of H' is defined for each state of length at most $i-1$ and assume that for each state q of H of length at most i there is exactly one state q' of H' in $sample^{-1}(q)$. Let q be a state of H of length i and let q' be the state of $sample^{-1}(q)$. Observe from the definition of *sample* that the length of q' is i. Let $(lstate(q), \mathcal{P})$ be the combined transition of M that corresponds to tr_q^H. For each pair (a, s) of Ω, if a is a discrete action, then let $\mathcal{P}_{(a,s')}$ be $\mathcal{D}((a, s'))$; if a is a time-passage action, then let $\mathcal{P}_{(a,s')}$ be $\mathcal{D}(w_{a,s'})$, where $w_{a,s'} \in trajectories(M, s, a, s')$. Let $\mathcal{P}' = \sum_{(a,s)\in\Omega} P[(a,s)]\mathcal{P}_{(a,s)}$. Then, $(lstate(q), \mathcal{P}')$ is a time-enriched transition of M. Let $tr_{q'}^{H'}$ be $(q', q' \frown \mathcal{P}')$. Then, $tr_{q'}^{H'}$ is a legal transition for H'. Moreover, from the definition of $\mathcal{P}'$, each state of $\mathcal{P}_q^H$ is the sampling of exactly one state of $\mathcal{P}_{q'}^{H'}$, and, vice versa, the sample of each state of $\mathcal{P}_{q'}^{H'}$ is a state of $\mathcal{P}_q^H$. ■

Probabilistic Time-Enriched Executions versus Probabilistic Timed Executions

We define a function *t-sample* that, given a probabilistic time-enriched execution fragment H of M, builds a probabilistic timed execution H' as follows.

$$\begin{aligned} \mathit{states}(H') \quad = \quad & \{\mathit{t\text{-}exec}(q_0^H)\} \cup \\ & \{q \in \Omega_{\mathit{t\text{-}exec}(H)} \mid q \text{ contains finitely many actions}\} \cup \\ & \{q \in \mathit{t\text{-}frag}^*(M) \mid \mathit{ltraj}(q) \text{ is a } [0,0]\text{-trajectory and } \exists_{q' \in \Omega_{\mathit{t\text{-}exec}(H)}} q \leq q'\}. \end{aligned} \tag{9.26}$$

The start state of H' is $\mathit{t\text{-}exec}(q_0^H)$, and for each state q of H' the transition enabled from q is $(q, \mathit{truncate}_q(\mathit{t\text{-}exec}(\mathcal{P}_H)|C_q))$.

Proposition 9.3.8 *t-sample*(H) *is a probabilistic timed execution fragment of* M*.*

Proof. We need to show that for each state q of H' that enables some transition there is a probabilistic time-enriched execution fragment H_q of M starting from $\mathit{lstate}(q)$ such that $\mathcal{P}_q^H = \mathit{truncate}_{\mathit{lstate}(q)}(\mathit{t\text{-}exec}(\mathcal{P}_{H_q}))$.

Let $q_1, q_2, \ldots$ be an enumeration of the states q' of H such that $\mathit{t\text{-}exec}(q') = q$, and for each i let p_i denote $P_H[C_{q_i}]$. Observe that, since q ends with the occurrence of a discrete action, for each state q'' of H such that $q' \leq \mathit{t\text{-}exec}(q'')$ there is an i such that $q_i \leq q''$. Define H_q as follows.

$$\mathit{states}(H_q) \triangleq \bigcup_i \mathit{states}(H \triangleright q_i). \tag{9.27}$$

For each state q' of H_q, let

$$\mathit{tr}_{q'}^{H_q} \triangleq \frac{\sum_{i | q' \in \mathit{states}(H \triangleright q_i)} P_H[C_{q_i \frown q'}](\mathit{tr}^H_{q_i \frown q'} \triangleright q_i)}{\sum_{i | q' \in \mathit{states}(H \triangleright q_i)} P_H[C_{q_i \frown q'}]}. \tag{9.28}$$

Then, it is enough to prove that

$$q \frown \mathit{t\text{-}exec}(\mathcal{P}_{H_q}) = \mathit{t\text{-}exec}(\mathcal{P}_H)|C_q. \tag{9.29}$$

Before proving (9.29), we show the following property: for each state q' of H_q,

$$P_{H_q}[C_{q'}] = \frac{\sum_{i | q' \in \mathit{states}(H \triangleright q_i)} P_H[C_{q_i \frown q'}]}{\sum_i P_H[C_{q_i}]}. \tag{9.30}$$

This follows easily by induction using Equation (9.28) for the inductive step. The denominator is necessary for the base case to work.

We now turn to Equation (9.29). Consider an extended timed execution fragment α of M, and distinguish the following two cases.

1. α does not end with an open trajectory.

   Suppose that $\alpha \in \Omega_{\mathit{t\text{-}exec}(\mathcal{P}_H)|C_q}$. Then, from the definition of $\mathit{t\text{-}exec}()$ and of the conditional operation, $q \leq \alpha$ and there is a time-enriched execution α' of Ω_H such that $\mathit{t\text{-}exec}(\alpha') = \alpha$. This means that there is a time-enriched execution α' of Ω_H such that $\mathit{t\text{-}exec}(\alpha') = \alpha$ and there is a state q_i of H such that $q_i \leq \alpha'$. From the construction of H_q, each prefix of α' is a state of H_q, and thus $\alpha' \in \Omega_{\mathit{t\text{-}exec}(H_q)}$. The argument can be reversed.

2. α ends with an open trajectory.

   Suppose that $\alpha \in \Omega_{t\text{-}exec(\mathcal{P}_H)|C_q}$. Then, from the definition of *t-exec*() and of the conditional operation, $q \le \alpha$ and for each finite prefix α' of α there is a timed execution α'' of $t\text{-}exec(\Omega_H)$ such that $\alpha' \le \alpha''$. It is sufficient to show that for each finite prefix α' of α there is a timed execution α''_q of $t\text{-}exec(\Omega_{H_q})$ such that $\alpha' \le (q \frown \alpha''_q)$. Consider a prefix α' of α, and let α'' be an element of $t\text{-}exec(\Omega_H)$ such that $\alpha' \le \alpha''$. Then there is a time-enriched execution α''' of Ω_H such that $\alpha' \le t\text{-}exec(\alpha''')$, which means that there is a finite prefix α'''' of α''' such that $\alpha' \le t\text{-}exec(\alpha''')$ and $q \le t\text{-}exec(\alpha''')$. Let q_i be the prefix of α''''. We know that such prefix exists. Then, from the definition of H_q, $\alpha''''\triangleright q_i$ is a state of H_q, and thus there is a time-enriched execution α'_q of Ω_{H_q} such that $\alpha' \le (q \frown t\text{-}exec(\alpha'_q))$. Moreover, $t\text{-}exec(\alpha'_q) \in t\text{-}exec(\mathcal{P}_{H_q})$, which is sufficient to conclude. The argument can be reversed.

Finally, we need to show that $P_{t\text{-}exec(\mathcal{P}_H)|C_q}$ and $P_{t\text{-}exec(\mathcal{P}_{H_q})}$ coincide on the cones of their sample spaces. Thus, consider a finite timed execution fragment α of M. From the definition of *t-exec*(),

$$P_{t\text{-}exec(\mathcal{P}_{H_q})}[C_\alpha] = \sum_{q' \in min(\{q' \in states(H_q) | \alpha \le t\text{-}exec(q')\})} P_{H_q}[C_{q'}]. \tag{9.31}$$

From (9.30),

$$P_{t\text{-}exec(\mathcal{P}_{H_q})}[C_\alpha] = \sum_{q' \in min(\{q' \in states(H_q) | \alpha \le t\text{-}exec(q')\})} \frac{\sum_{i | q' \in states(H \triangleright q_i)} P_H[C_{q_i \frown q'}]}{\sum_i P_H[C_{q_i}]}. \tag{9.32}$$

From the definition of the states of H_q, (9.32) can be rewritten into

$$P_{t\text{-}exec(\mathcal{P}_{H_q})}[C_\alpha] = \frac{\sum_i \sum_{q' \in min(\{q' \in states(H \triangleright q_i) | q \frown \alpha \le t\text{-}exec(q_i \frown q')\})} P_H[C_{q_i \frown q'}]}{\sum_i P_H[C_{q_i}]}. \tag{9.33}$$

By simplifying the concatenations we obtain

$$P_{t\text{-}exec(\mathcal{P}_{H_q})}[C_\alpha] = \frac{\sum_{q' \in min(\{q' \in states(H) | q \frown \alpha \le t\text{-}exec(q')\})} P_H[C_{q'}]}{\sum_i P_H[C_{q_i}]}. \tag{9.34}$$

From the definition of *t-exec*(), the definition of a conditional space, and the definition of the q_i's,

$$P_{t\text{-}exec(\mathcal{P}_H)|C_q}[C_\alpha] = \frac{\sum_{q' \in min(\{q' \in states(H) | q \frown \alpha \le t\text{-}exec(q')\})} P_H[C_{q'}]}{\sum_i P_H[C_{q_i}]}. \tag{9.35}$$

Since the right sides of Equations (9.34) and (9.35) are the same, we conclude that

$$P_{t\text{-}exec(\mathcal{P}_{H_q})}[C_\alpha] = P_{t\text{-}exec(\mathcal{P}_H)|C_q}[C_{q \frown \alpha}]. \tag{9.36}$$

This completes the proof. ■

Conversely, we show that every probabilistic timed execution of M is sampled by some probabilistic time-enriched execution of M. Let H be a probabilistic timed execution of M. Then, build H' as follows. Let H_0 be a probabilistic timed execution consisting of a single state that is t-sampled by q_0^H, i.e., $\textit{t-sample}(q_0^{H_0}) = q_0^H$. Strictly speaking H_0 is not a probabilistic timed execution because $q_0^{H_0}$ should enable a transition in general. Suppose now that H_i is defined. Then build H_{i+1} be extending the transition relation of H_i from all the states of H_i that do not end in δ and do not have any outgoing transition as follows. Consider a state q of H_i that do not end in δ and do not have any outgoing transition, and let q' be the state of H such that $\textit{t-exec}(q) = q'$ (our construction ensures that there is always such a state since q ends with a $[0,0]$-trajectory). From the definition of a probabilistic timed execution fragment, there is a probabilistic time-enriched execution fragment $H_{q'}$ of M starting from $\textit{lstate}(q')$ such that $\mathcal{P}_{q'}^H = \textit{truncate}_{\textit{lstate}(q')}(\textit{t-exec}(\mathcal{P}_{H_{q'}}))$. Let $H'_{q'}$ be obtained from $H_{q'}$ by removing all the transitions from states where an action has occurred and by removing all the states that become unreachable. Then, extend H_i from q' with $q' \frown H'_{q'}$, i.e., $H_{i+1} \triangleright q' = H'_{q'}$.

Then the states of H' are the union of the states of the H_i's, the start state of H' is $q_0^{H_0}$, and for each state q of H', if q is a state of H_i, then $tr_q^{H'} = tr_q^{H_{i+1}}$.

Proposition 9.3.9 *$\textit{t-sample}(H') = H$.*

Proof. We prove that $\mathcal{P}_H = \textit{t-exec}(\mathcal{P}_{H'})$. Then the equality between $\textit{t-sample}(H')$ and H follows by induction after observing that $\textit{t-sample}(H')$ and H have the same start state and that for each state q, $\textit{step}_q^{\textit{t-sample}(H')} = (q, \textit{truncate}_q(\textit{t-exec}(\mathcal{P}_{H'})|C_q))$, and that $\textit{step}_q^H = (q, \textit{truncate}_q(\mathcal{P}_H|C_q))$.

For the sample spaces, consider an element α of Ω_H. Then, by definition of Ω_H, there is an execution $\alpha_0\alpha_1\cdots$ of H such that $\lim_i \alpha_i = \alpha$, and such that either α is not a finite execution, or the last element of α ends in δ. We distinguish two cases.

1. α is either an infinite sequence or a finite sequence $\alpha_0\alpha_2\cdots\alpha_n$ where α_n ends with δ.

   From the definition of the transition relation of H', there is a sequence of extended time-enriched execution fragments $q_0, q_1, \ldots$ such that for each i $\alpha_i = \textit{t-exec}(q_0 \frown \cdots \frown q_i)$, $q_0 \frown q_1 \frown \cdots$ is an element of $\Omega_{H'}$, and $\textit{t-exec}(q_0 \frown q_1 \frown \cdots) = \alpha$. Thus, $\alpha \in \Omega_{\textit{t-exec}(H')}$. The converse argument is a reversal of the argument above.

2. $\alpha = \alpha_0\alpha_2\cdots\alpha_n$ where α_n ends with an open trajectory.

   From the definition of the transition relation of H', there is a sequence of extended time-enriched execution fragments $q_0, q_1, \ldots, q_{n-1}$ such that for each $i \leq n-1$ $\alpha_i = \textit{t-exec}(q_0 \frown \cdots \frown q_i)$ and $q_0 \frown \cdots \frown q_i$ is a state of H'. Furthermore, for each finite prefix α' of α there is a time-enriched execution fragment q_n such that $\alpha' \leq \textit{t-exec}(q_0 \frown \cdots \frown q_n)$ and $q_0 \frown \cdots \frown q_{n-1} \frown q_n$ is an element of $\Omega_{H'}$. This means that for each finite prefix α' of α there is an element α'' of $\textit{t-exec}(\Omega_{H'})$ such that $\alpha' \leq \alpha''$, and thus $\alpha \in \Omega_{\textit{t-exec}(\mathcal{P}_{H'})}$. The argument can be reversed.

Consider now a cone C_α. From the definition of $\textit{t-exec}()$,

$$P_{\textit{t-exec}(H')}[C_\alpha] = \sum_{q \in \min(\{q \in \textit{states}(H') | \alpha \leq \textit{t-exec}(q)\})} P_{H'}[C_q]. \tag{9.37}$$

If C_α is not empty, then $\alpha = \alpha_1 \cdots \alpha_n$, where $\alpha_n = \alpha$, $\alpha_0 \cdots \alpha_{n-1}$ is an execution of H, and there is a α'_n such that $\alpha_n \leq \alpha'_n$ and $\alpha_1 \cdots \alpha'_n$ is an execution of H. We show by induction on

n that

$$P_H[C_{\alpha_n}] = \sum_{q \in min(\{q \in states(H') \mid \alpha \leq t\text{-}exec(q)\})} P_{H'}[C_q]. \tag{9.38}$$

The base case is trivial since C_{α_0} denotes the whole sample space. For the inductive case, from the definition of the probability of a cone,

$$P_H[C_{\alpha_n}] = P_H[C_{\alpha_{n-1}}]P^H_{\alpha_{n-1}}[C_{\alpha_n}]. \tag{9.39}$$

From the definition of the transition relation of H,

$$P^H_{\alpha_{n-1}}[C_{\alpha_n}] = \frac{\sum_{q \in states(H') \mid t\text{-}exec(q)=\alpha_{n-1}} P_{H'}[C_q] P_{t\text{-}exec(H' \triangleright q)}[C_{\alpha_n \triangleright \alpha_{n-1}}]}{\sum_{q \in states(H') \mid t\text{-}exec(q)=\alpha_{n-1}} P_{H'}[C_q]}, \tag{9.40}$$

where

$$P_{t\text{-}exec(H' \triangleright q)}[C_{\alpha_n \triangleright \alpha_{n-1}}] = \sum_{q' \in min(\{q' \in states(H' \triangleright q) \mid \alpha_n \leq t\text{-}exec(q \frown q')\})} P_{H' \triangleright q}[C_{q'}]. \tag{9.41}$$

Since α_{n-1} is a state of H, the last trajectory of α_{n-1} has domain $[0,0]$, and the set $\{q \in states(H') \mid t\text{-}exec(q) = \alpha_{n-1}\}$ is a set of minimal states. Thus, by substituting (9.41) in (9.40), simplifying the numerator of (9.40), we obtain

$$P_{t\text{-}exec(H' \triangleright q)}[C_{\alpha_n \triangleright \alpha_{n-1}}] = \frac{\sum_{q' \in min(\{q' \in states(H') \mid \alpha_n \leq t\text{-}exec(q')\})} P_{H'}[C_{q'}]}{\sum_{q \in states(H') \mid t\text{-}exec(q)=\alpha_{n-1}} P_{H'}[C_q]}. \tag{9.42}$$

By substituting (9.42) in (9.39), using induction and simplifying algebraically, we get (9.38). ■

Equivalent Probabilistic Time-Enriched Executions

It is possible to define an equivalence relation on probabilistic time-enriched executions that captures exactly the probabilistic timed executions that they represent.

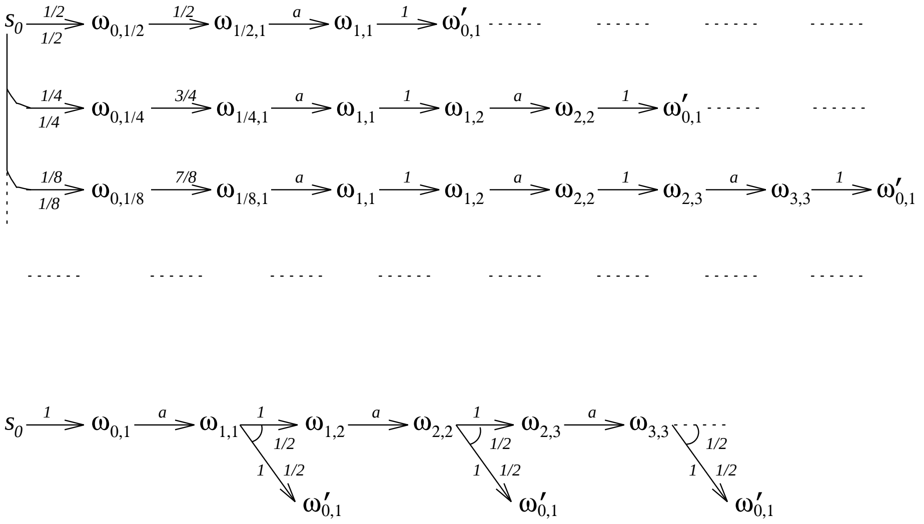
Let H_1 and H_2 be two probabilistic time-enriched execution fragments of a probabilistic timed automaton M. Then $t\text{-}exec(\mathcal{P}_{H_1})$ and $t\text{-}exec(\mathcal{P}_{H_2})$ are said to be *equivalent*, denoted by $t\text{-}exec(\mathcal{P}_{H_1}) \equiv t\text{-}exec(\mathcal{P}_{H_2})$, iff

1. for each timed extended execution fragment α of M that does not contain infinitely many discrete actions, $\alpha \in \Omega_{t\text{-}exec(\mathcal{P}_{H_1})}$ iff $\alpha \in \Omega_{t\text{-}exec(\mathcal{P}_{H_2})}$;

2. for each finite timed extended execution fragment α of M, $P_{t\text{-}exec(\mathcal{P}_{H_1})}[C_\alpha] = P_{t\text{-}exec(\mathcal{P}_{H_2})}[C_\alpha]$.

H_1 and H_2 are said to be *equivalent*, denoted by $H_1 \equiv H_2$, iff $t\text{-}exec(q_0^{H_1}) = t\text{-}exec(q_0^{H_2})$ and $t\text{-}exec(\mathcal{P}_{H_1}) \equiv t\text{-}exec(\mathcal{P}_{H_2})$.

Example 9.3.3 (Two equivalent probabilistic time-enriched executions) In the definition above we do not require the sample spaces of the given probabilistic time-enriched execution fragments to contain the same timed executions with infinitely many discrete actions. Figure 9-2 shows an example of two probabilistic time-enriched executions whose corresponding sample spaces differ from a timed execution with infinitely many discrete actions and such that



s_0
1/2, 1/2, $\omega_{0,1/2}$, 1/2, $\omega_{1/2,1}$, a, $\omega_{1,1}$, 1, $\omega'_{0,1}$
1/4, 1/4, $\omega_{0,1/4}$, 3/4, $\omega_{1/4,1}$, a, $\omega_{1,1}$, 1, $\omega_{1,2}$, a, $\omega_{2,2}$, 1, $\omega'_{0,1}$
1/8, 1/8, $\omega_{0,1/8}$, 7/8, $\omega_{1/8,1}$, a, $\omega_{1,1}$, 1, $\omega_{1,2}$, a, $\omega_{2,2}$, 1, $\omega_{2,3}$, a, $\omega_{3,3}$, 1, $\omega'_{0,1}$
s_0, 1, $\omega_{0,1}$, a, $\omega_{1,1}$, 1, $\omega_{1,2}$, a, $\omega_{2,2}$, 1, $\omega_{2,3}$, a, $\omega_{3,3}$
1/2, 1, 1/2, $\omega'_{0,1}$
1/2, 1, 1/2, $\omega'_{0,1}$
1/2, 1, 1/2, $\omega'_{0,1}$

Figure 9-2: Probabilistic time-enriched executions that represent the same probabilistic timed execution.

t-sample() gives the same probabilistic timed execution. The important aspect of this example is that in the upper probabilistic time-enriched execution the explicit time-passage actions are used to let 1 time unit elapse in infinitely many different ways. However, the trajectory that is spanned before the first occurrence of action a is always the same. Observe that the fact that the two probabilistic time-enriched executions of Figure 9-2 represent the same structure is not a consequence of the limit closure of the sample space of *t-exec*(), since $\textit{t-exec}(\Omega_{H_1})$ and $\textit{t-exec}(\Omega_{H_2})$ do not differ in timed executions that end with an open trajectory. Rather, by analyzing this example again in the context of parallel composition we will discover the reason for our definition of *t-exec*() (cf. Example 9.5.1). ■

The rest of this section is dedicated to showing that $\equiv$ characterizes the probabilistic timed executions represented by probabilistic time-enriched executions. We do it by showing two results: the first result says that two equivalent probabilistic time-enriched executions describe the same probabilistic timed execution, and the second result says that for each probabilistic time-enriched execution H, $\mathcal{P}_{\textit{t-sample}(H)} \equiv \textit{t-exec}(\mathcal{P}_H)$.

Proposition 9.3.10 *If* $\textit{t-exec}(H_1) \equiv \textit{t-exec}(H_2)$, *then* $\textit{t-sample}(H_1) = \textit{t-sample}(H_2)$.

Proof. Let $q \in \textit{states}(\textit{t-sample}(H_1))$. If $q = \textit{t-exec}(q_0^{H_1})$ or $q \in \Omega_{\textit{t-exec}(H_1)}$ and contains finitely many discrete actions, then $q \in \textit{states}(\textit{t-sample}(H_2))$ trivially. Thus, suppose that $\textit{ltraj}(q)$ is a [0,0]-trajectory and that there is a $q' \in \Omega_{\textit{t-exec}(H_1)}$ such that $q \leq q'$. Then, $P_{\textit{t-exec}(H_1)}[C_q] > 0$, and, since $\textit{t-exec}(H_1) = \textit{t-exec}(H_2)$, $P_{\textit{t-exec}(H_2)}[C_q] > 0$. Thus, there is a $q'' \in \Omega_{\textit{t-exec}(H_2)}$ such that $q \leq q''$, which means that $q \in \textit{states}(\textit{t-sample}(H_2))$. The converse argument is identical.

Consider now a state q of $\textit{t-sample}(H_1)$ and $\textit{t-sample}(H_2)$. We need to show that $tr_q^{\textit{t-sample}(H_1)}$ and $tr_q^{\textit{t-sample}(H_2)}$ are the same transition. From the definition of *t-sample*(), it is enough to show that $\textit{truncate}_q(\textit{t-exec}(\mathcal{P}_{H_1})|C_q) = \textit{truncate}_q(\textit{t-exec}(\mathcal{P}_{H_2})|C_q)$. Since $\textit{t-exec}(\mathcal{P}_{H_1}) \equiv \textit{t-exec}(\mathcal{P}_{H_2})$, a direct analysis of the definition of $\textit{t-exec}()$ shows that $\textit{t-exec}(\mathcal{P}_{H_1})|C_q \equiv \textit{t-exec}(\mathcal{P}_{H_2})|C_q$. The truncation operation is independent of the elements of Ω that contains infinitely many discrete actions, and thus $\Omega_{truncate_q(\textit{t-exec}(\mathcal{P}_{H_1})|C_q)} = \Omega_{truncate_q(\textit{t-exec}(\mathcal{P}_{H_2})|C_q)}$. Furthermore, directly from the definition of $\equiv$, $P_{truncate_q(\textit{t-exec}(\mathcal{P}_{H_1})|C_q)}$ and $P_{truncate_q(\textit{t-exec}(\mathcal{P}_{H_2})|C_q)}$ coincide on the cones, and thus $truncate_q(\textit{t-exec}(\mathcal{P}_{H_1})|C_q) = truncate_q(\textit{t-exec}(\mathcal{P}_{H_2})|C_q)$. ∎

Proposition 9.3.11 *Let H be a probabilistic time-enriched execution of a probabilistic timed automaton M. Then, $\mathcal{P}_{\textit{t-sample}(H)} \equiv \textit{t-exec}(\mathcal{P}_H)$.*

Proof. Consider a finite timed execution α of M. We prove the proposition in three steps.

1. For each finite timed extended execution α of M, there is a timed extended execution α' of $\Omega_{\textit{t-sample}(H)}$ such that $\alpha \leq \alpha'$ iff there is a timed extended execution α'' of $\Omega_{\textit{t-exec}(\mathcal{P}_H)}$ such that $\alpha \leq \alpha''$.

   Let $\alpha' \in \Omega_{\textit{t-sample}(H)}$ such that $\alpha \leq \alpha'$. Then there is a complete execution $q_0 q_1 \cdots$ of $\textit{t-sample}(H)$ such that $lim_i q_i = \alpha'$. In particular, there is a value n such that $\alpha \leq q_n$. From the definition of the transition relation of $\textit{t-sample}(H)$, $P_{\textit{t-exec}(H)}[C_{q_n}] > 0$, and thus there is a timed execution α'' of $\Omega_{\textit{t-exec}(\mathcal{P}_H)}$ such that $q_n \leq \alpha''$, which means that $\alpha \leq \alpha''$. Conversely, suppose that there is a timed execution α'' of $\Omega_{\textit{t-exec}(\mathcal{P}_H)}$ such that $\alpha \leq \alpha''$. If α'' contains finitely many actions, then $\alpha'' \in \Omega_{\textit{t-sample}(H)}$ by definition. Otherwise, there is a finite prefix α''' of α'' such that $\alpha \leq \alpha'''$ and the last trajectory of α''' has domain $[0,0]$. From the definition of $\textit{t-sample}(H)$, α''' is a state of $\textit{t-sample}(H)$, and thus there is a timed execution α' of $\Omega_{\textit{t-sample}(H)}$ such that $\alpha''' \leq \alpha'$, which means that $\alpha \leq \alpha'$.

2. For each timed extended execution fragment α of M that does not contain infinitely many discrete actions, $\alpha \in \Omega_{\textit{t-sample}(H)}$ iff $\alpha \in \Omega_{\textit{t-exec}(\mathcal{P}_H)}$.

   Let α be a timed extended execution of M that does not contain infinitely many discrete actions, and suppose that $\alpha \in \Omega_{\textit{t-sample}(H)}$. If α ends with δ, then Item 1 is sufficient to conclude that $\alpha \in \Omega_{\textit{t-exec}(\mathcal{P}_H)}$. If α does not end with δ, then there is a finite execution $q_0 q_1 \cdots q_n$ of $\textit{t-sample}(H)$ such that q_n ends with a right-open trajectory. From the definition of the transition relation of $\textit{t-sample}(H)$, $q_n \in truncate_{q_{n-1}}(\textit{t-exec}(\mathcal{P}_H)|C_{q_{n-1}})$. Since q_n ends with an open trajectory, $q_n \in \Omega_{\textit{t-exec}(\mathcal{P}_H)}$, i.e., $\alpha \in \Omega_{\textit{t-exec}(\mathcal{P}_H)}$.

   Conversely, suppose that $\alpha \in \Omega_{\textit{t-exec}(\mathcal{P}_H)}$. If α ends with δ, then Item 1 is sufficient to conclude that $\alpha \in \Omega_{\textit{t-sample}(H)}$. If α does not end with δ, then there is a finite prefix α' of α such that $\alpha \triangleright \alpha'$ does not contain any action, and either α' is the start state of $\textit{t-sample}(H)$, or the last trajectory of α' has domain $[0,0]$. Thus, from the definition of $\textit{t-sample}()$, α' is a state of $\textit{t-sample}(H)$. From the definition of $truncate$, $\alpha \in truncate_{\alpha'}(\textit{t-exec}(\mathcal{P}_H)|C_{\alpha'})$, and thus, from the definition of the transition relation of $\textit{t-sample}(H)$, $\alpha \in \Omega_{\alpha'}^{\textit{t-sample}(H)}$. Since α ends with an open trajectory, $\alpha \in \Omega_{\textit{t-sample}(H)}$.

3. For each finite timed extended execution fragment α of M, $P_{\textit{t-sample}(H)}[C_\alpha] = P_{\textit{t-exec}(\mathcal{P}_H)}[C_\alpha]$.

   Let α be a finite timed execution. From Item 1, $C_\alpha^{\textit{t-sample}(H)} = \emptyset$ iff $C_\alpha^{\textit{t-exec}(\mathcal{P}_H)} = \emptyset$. Suppose that $C_\alpha^{\textit{t-sample}(H)}$ is not empty. Then there is an execution of $\textit{t-sample}(H)$,

$\alpha_0\alpha_1\cdots\alpha_{n-1}\alpha_n$ such that $\alpha_{n-1} < \alpha \leq \alpha_n$. From the definition of the probability of a cone,

$$P_{\textit{t-sample}(H)}[C_\alpha] = P_{\alpha_0}[C_{\alpha_1}]P_{\alpha_1}[C_{\alpha_2}]\cdots P_{\alpha_{n-2}}[C_{\alpha_{n-1}}]P_{\alpha_{n-1}}[C_\alpha]. \tag{9.43}$$

From the definition of *t-sample*(H), for each $i < n$

$$P_{\alpha_i}[C_{\alpha_{i+1}}] = P_{\textit{t-exec}(H)|C_{\alpha_i}}[C_{\alpha_{i+1}}]. \tag{9.44}$$

Thus, by substituting (9.44) in (9.43) and simplifying, we obtain

$$P_{\textit{t-sample}(H)}[C_\alpha] = P_{\textit{t-exec}(H)}[C_\alpha]. \tag{9.45}$$

This completes the proof. ■

9.4 Moves

In the non-timed framework we have introduced the notion of a weak transition to abstract from internal computation. Informally, a weak transition is obtained by concatenating several internal and external transitions so that overall the system emulates a unique transition labeled with at most one external action. In the timed framework, due to the presence of explicit time-passage actions, it may be the case that some time t cannot elapse without performing some internal transitions in the middle. This problem becomes more evident when we extend the simulation relations to the timed framework (cf. Chapter 12). For this reason we introduce the concept of a *m*ove, which extends weak transitions and abstracts from internal transitions interleaved with time-passage transitions..

Let M is a probabilistic timed automaton, s be a state of M, $\mathcal{P}$ be a discrete probability distribution over states of M, and a be an action of M or the value 0. If a is a visible action of M then we use the expression $s \overset{a}{\rightsquigarrow} \mathcal{P}$ to denote $s \overset{a}{\Longrightarrow} \mathcal{P}$; if $a = 0$, then we use the expression $s \overset{0}{\rightsquigarrow} \mathcal{P}$ to denote $s \rightsquigarrow \mathcal{P}$, which is the same as $s \Longrightarrow \mathcal{P}$; if a is a time-passage action, i.e., $a = d$ for some $d \in \Re^+$, then we use the expression $s \overset{d}{\rightsquigarrow} \mathcal{P}$ to denote that $\mathcal{P}$ is reached from s by means of several internal and time-passage transitions so that in each situation time d has elapsed. Formally, $s \overset{d}{\rightsquigarrow} \mathcal{P}$ 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 $\alpha\delta \in \Omega_H$, *t-trace*$(\alpha)$ = *t-trace*(a);
4. $\mathcal{P}$ = *lstate*(δ-*strip*$(\mathcal{P}_H)$), where δ-*strip*$(\mathcal{P}_H)$ is the probability space $\mathcal{P}'$ such that $\Omega' = \{\alpha \mid \alpha\delta \in \Omega_H\}$, and for each $\alpha \in \Omega'$, $P'[\alpha] = P_H[C_{\alpha\delta}]$;

The notion of a generator for a weak transition can be extended to moves in a straightforward way.

9.5 Parallel Composition

The parallel composition operator for probabilistic timed automata is exactly the same as the parallel composition operator for probabilistic automata. Thus, we omit the formal definition. According to the definition of the transition relation of $M_1\|M_2$, M_1 and M_2 synchronize on all their time-passage transitions, and thus time advances always at the same speed in M_1 and M_2.

The definition of a projection of a probabilistic time-enriched execution is the same as the definition of a projection of a probabilistic execution, except that the states of a probabilistic time-enriched execution fragment are time-enriched execution fragments rather than ordinary execution fragments. Thus, we need to extend the definition of a projection to time-enriched execution fragments and time-enriched transitions.

Let M be $M_1\|M_2$, and let α be a time-enriched execution of M. The projection of α onto M_i, $i = 1, 2$, is the sequence obtained from α by projecting the codomain of each trajectory onto M_i, by removing all the actions not in $acts(M_i)$, and by concatenating all the trajectories whose intermediate actions are removed. It is straightforward to check that α is a time-enriched execution of M_i.

Let H be a probabilistic time-enriched execution of M, and 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 $\delta\lceil M_i = \delta$. The projection of tr onto M_i, denoted by $tr\lceil M_i$, is the pair $(q\lceil M_i, \mathcal{P}\lceil M_i)$.

Proposition 9.5.1 *Let $M = M_1\|M_2$, and let H be a probabilistic time-enriched execution fragment of M. Then $H\lceil M_1 \in$ t-prexec(M_1) and $H\lceil M_2 \in$ t-prexec(M_2).*

Proof. The structure of the proof is the same as the proof of Proposition 4.3.4. This time it is necessary to observe that for each state q of H the transition $(tr^H_{q'} \upharpoonright acts(M_1))\lceil M_1$ is generated by a time-enriched transition of M_i. ■

Proposition 9.5.2 *Let $M = M_1\|M_2$, and let H be a probabilistic time-enriched execution fragment of M. Let H_i be $H\lceil M_i$, $i = 1, 2$. Let q be a state of H_i. Then,*

$$P_{H_i}[C_q] = \sum_{q' \in min(q\rceil H)} P_H[C_{q'}]. \tag{9.46}$$

Proof. This proof has the same structure as the proof of Proposition 4.3.5. ■

In the rest of this section we extend the results of Section 9.3.3 to account for parallel composition. We show that *sample* commutes with projections and that the projections of two equivalent probabilistic time-enriched executions are equivalent. The first result guarantees that *sample* and projection are well defined for probabilistic time-enriched executions; the second result allows us to define indirectly a projection operator on probabilistic timed executions: namely, given a probabilistic timed execution H of $M_1\|M_2$, let H' be any probabilistic time-enriched execution of $M_1\|M_2$ such that *t-sample*$(H') = H$. Then, $H\lceil M_i$ is defined to be *t-sample*$(H'\lceil M_i)$. Before proving these two results, we show why in the definition of *t-exec*() we force probabilistic time-enriched executions like those of Figure 9-1 to be mapped to the same structure (cf. Example 9.3.2).

Example 9.5.1 (Reason for the definition of *t-exec*) We have already seen that the probabilistic time-enriched executions of Figure 9-2 are t-samples of the same probabilistic timed execution. Suppose now the probabilistic time-enriched executions of Figure 9-2 to be probabilistic time-enriched executions of the parallel composition of two probabilistic timed automata M_1 and M_2, and suppose that a is an action of M_2 only. By projecting the probabilistic time-enriched executions of Figure 9-2 onto M_1 we obtain two probabilistic time-enriched executions like those of Figure 9-1, which must denote the same probabilistic timed execution if we want *t-sample* to be preserved by the projection operation. ■

Proposition 9.5.3 *Let M be $M_1 \| M_2$, and let H be a probabilistic time-enriched execution of M. Then, $sample(H\lceil M_i) = sample(H)\lceil M_i$.*

Proof. Since the sampling function commutes with the projection function, $sample(H\lceil M_i)$ and $sample(H)\lceil M_i$ have the same states.

For convenience, denote $sample(H)$ by H'. Let q be one of the states of $sample(H)\lceil M_i$. Below we show that the equation for the transition leaving from q in $sample(H)\lceil M_i$ and the equation for the transition leaving from q in $sample(H\lceil M_i)$ denote the same transition. This is sufficient to show that $sample(H)\lceil M_i$ and $sample(H\lceil M_i)$ have the same transition relation. We use implicitly the fact that the projection onto M_i distributes over the sum of transitions restricted to $acts(M_i)$.

From (9.25), Proposition 4.3.2, and an algebraic simplification, the expression

$$\sum_{q' \in q\rceil H'} \bar{p}_{q'}^{q\rceil H'} P_{q'}^{H'}[acts(M_i)](tr_{q'}^{H'} \upharpoonright acts(M_i))\lceil M_i \tag{9.47}$$

can be rewritten into

$$\sum_{q' \in q\rceil H'} \sum_{q'' \in sample^{-1}(q')} \bar{p}_{q'}^{q\rceil H'} \bar{p}_{q''}^{sample^{-1}(q')} sample(tr_{q''}^{H} \upharpoonright acts(M_i))\lceil M_i, \tag{9.48}$$

which becomes

$$\sum_{q'' \in sample^{-1}(q\rceil H')} \bar{p}_{sample(q'')}^{q\rceil H'} \bar{p}_{q''}^{sample^{-1}(sample(q''))} sample(tr_{q''}^{H} \upharpoonright acts(M_i))\lceil M_i, \tag{9.49}$$

after grouping the two sums.

Denote $H\lceil M_i$ by H''. From (4.22), Proposition 4.3.2, and an algebraic simplification,

$$\sum_{q' \in sample^{-1}(q)} \bar{p}_{q'}^{sample^{-1}(q)} sample(tr_{q'}^{H''}) \tag{9.50}$$

can be rewritten into

$$\sum_{q' \in sample^{-1}(q)} \sum_{q'' \in q'\rceil H} \bar{p}_{q'}^{sample^{-1}(q)} \bar{p}_{q''}^{q'\rceil H} P_{q''}^{H}[acts(M_i)] sample(tr_{q''}^{H} \upharpoonright acts(M_i))\lceil M_i, \tag{9.51}$$

which becomes

$$\sum_{q'' \in (sample^{-1}(q))\rceil H} \bar{p}_{q''\lceil M_i}^{sample^{-1}(q)} \bar{p}_{q''}^{(q''\lceil M_i)\rceil H} P_{q''}^{H}[acts(M_i)] sample(tr_{q''}^{H} \upharpoonright acts(M_i))\lceil M_i \tag{9.52}$$

after grouping the two sums.

From the commutativity of *sample* and projection, $sample^{-1}(q\rceil H') = sample^{-1}(q)\rceil H$. Thus, in order to show that (9.49) and (9.52) denote the same transition, it is sufficient to show that for each state q'' of $sample^{-1}(q\rceil H')$,

$$\bar{p}_{sample(q'')}^{q\rceil H'} \bar{p}_{q''}^{sample^{-1}(sample(q''))} = \bar{p}_{q''\lceil M_i}^{sample^{-1}(q)} \bar{p}_{q''}^{(q''\lceil M_i)\rceil H}. \tag{9.53}$$

By expanding the expressions above with their definitions, (9.53) becomes

$$\begin{aligned} &\frac{P_{H'}[C_{sample(q'')}]P_H[C_{q''}]}{(\sum_{\bar{q}'\in min(q\rceil H')} P_{H'}[C_{\bar{q}'}])(\sum_{\bar{q}''\in sample^{-1}(sample(q''))} P_H[C_{\bar{q}''}])} \\ &= \frac{P_{H''}[C_{q''\lceil M_i}]P_H[C_{q''}]}{(\sum_{\bar{q}'\in sample^{-1}(q)} P_{H''}[C_{\bar{q}'}])(\sum_{\bar{q}''\in min((q''\lceil M_i)\rceil H)} P_H[C_{\bar{q}''}])}. \end{aligned} \tag{9.54}$$

By simplifying common subexpressions, using Proposition 4.3.5, and observing that

$$P_{H'}[C_{sample(q'')}] = \sum_{\bar{q}''\in sample^{-1}(sample(q''))} P_H[C_{\bar{q}''}], \tag{9.55}$$

(we have verified properties like (9.55) several times) Equation (9.54) becomes

$$\sum_{\bar{q}'\in min(q\rceil H')} P_{H'}[C_{\bar{q}'}] = \sum_{\bar{q}'\in sample^{-1}(q)} P_{H''}[C_{\bar{q}'}], \tag{9.56}$$

which can be shown as follows:

$$\begin{aligned} &\sum_{\bar{q}'\in min(q\rceil H')} P_{H'}[C_{\bar{q}'}] \\ &= \sum_{\bar{q}'\in min(q\rceil H')} \sum_{q''\in sample^{-1}(\bar{q}')} P_H[C_{q''}] \\ &= \sum_{q''\in min(sample^{-1}(q\rceil H'))} P_H[C_{q''}] \\ &= \sum_{q''\in min((sample^{-1}(q))\rceil H)} P_H[C_{q''}] \\ &= \sum_{\bar{q}'\in sample^{-1}(q)} \sum_{q''\in min(\bar{q}'\rceil H)} P_H[C_{q''}] \\ &= \sum_{\bar{q}'\in sample^{-1}(q)} P_{H''}[C_{\bar{q}'}], \end{aligned}$$

where the first step follows from (9.55), the second and fourth steps follow from grouping and ungrouping sums, the third step follows from the commutativity of *sample* and projection, and the fifth step follows from Proposition 4.3.5. ■

Proposition 9.5.4 *Let H_1 and H_2 be two probabilistic time-enriched executions of $M_1\|M_2$. If $H_1 \equiv H_2$, then $H_1\lceil M_i \equiv H_2\lceil M_i$, $i = 1, 2$.*

Proof. We show first that $\textit{t-exec}(\mathcal{P}_{H_1\lceil M_i})$ and $\textit{t-exec}(\mathcal{P}_{H_2\lceil M_i})$ assign the same probabilities to the same cones; then we show that the sample spaces of $\textit{t-exec}(\mathcal{P}_{H_1\lceil M_i})$ and $\textit{t-exec}(\mathcal{P}_{H_2\lceil M_i})$ satisfy the condition for $\equiv$. This part of the proof relies on the way we have defined the sample spaces of the objects produced by $\textit{t-exec}()$. For the cones, we show that for each finite timed extended execution α of M_i,

$$P_{\textit{t-exec}(\mathcal{P}_{H_1\lceil M_i})}[C_\alpha] = \sum_{\alpha' \in \mathit{min}(\{\alpha' \in \textit{t-frag}^*_\delta(M_1 \| M_2) \mid \alpha = \alpha' \lceil M_i\})} P_{\textit{t-exec}(H_1)}[C_{\alpha'}]. \tag{9.57}$$

and

$$P_{\textit{t-exec}(\mathcal{P}_{H_2\lceil M_i})}[C_\alpha] = \sum_{\alpha' \in \mathit{min}(\{\alpha' \in \textit{t-frag}^*_\delta(M_1 \| M_2) \mid \alpha = \alpha' \lceil M_i\})} P_{\textit{t-exec}(H_2)}[C_{\alpha'}]. \tag{9.58}$$

Then, since $H_1 \equiv H_2$, we conclude that the right sides of (9.57) and (9.58) are equal, and thus, $H_1\lceil M_i \equiv H_2\lceil M_i$. We prove only (9.57); the proof for (9.58) is symmetric. From the definition of $\textit{t-exec}()$,

$$P_{\textit{t-exec}(\mathcal{P}_{H_1\lceil M_i})}[C_\alpha] = \sum_{q \in \mathit{min}(\{q \in \mathit{states}(H_1\lceil M_i) \mid \alpha \le \textit{t-exec}(q)\})} P_{H_1\lceil M_i}[C_q]. \tag{9.59}$$

From (4.31),

$$P_{\textit{t-exec}(\mathcal{P}_{H_1\lceil M_i})}[C_\alpha] = \sum_{q \in \mathit{min}(\{q \in \mathit{states}(H_1\lceil M_i) \mid \alpha \le \textit{t-exec}(q)\})} \left(\sum_{q' \in \mathit{min}(q \rceil H_1)} P_{H_1}[C_{q'}] \right). \tag{9.60}$$

Consider a state q of $\mathit{min}(\{q \in \mathit{states}(H_1\lceil M_i) \mid \alpha \le \textit{t-exec}(q)\})$ and a state q' of $\mathit{min}(q\rceil H_1)$. Then, from the definition of $\textit{t-exec}()$, there is at least one $\alpha' \in \textit{t-frag}^*_\delta(M_1\|M_2)$ such that $\alpha = \alpha'\lceil M_i$ and $q' \in \mathit{min}(\{q' \in \mathit{states}(H_1) \mid \alpha' \le \textit{t-exec}(q')\})$. Moreover, there is exactly one minimum α'. Conversely, consider one $\alpha' \in \mathit{min}(\{\alpha' \in \textit{t-frag}^*_\delta(M_1\|M_2) \mid \alpha = \alpha'\lceil M_i\})$, and consider a state q' of $\mathit{min}(\{q' \in \mathit{states}(H_1) \mid \alpha' \le \textit{t-exec}(q')\})$. Let $q = q'\lceil M_i$. Then, $q' \in \mathit{min}(q\rceil H_1)$ and q is a state of $\mathit{min}(\{q \in \mathit{states}(H_1\lceil M_i) \mid \alpha \le \textit{t-exec}(q)\})$. Thus, from (9.60) we obtain (9.57).

We now move to the sample spaces. Let α be an element of $\Omega_{\textit{t-exec}(\mathcal{P}_{H_1\lceil M_i})}$ that does not contain infinitely many discrete actions. If α ends with δ, then α is trivially an element of $\Omega_{\textit{t-exec}(\mathcal{P}_{H_2\lceil M_i})}$ since $P_{\textit{t-exec}(\mathcal{P}_{H_2\lceil M_i})}[C_\alpha] = P_{\textit{t-exec}(\mathcal{P}_{H_2\lceil M_i})}[C_\alpha] > 0$. Otherwise, α ends with an open trajectory. Then, from the definition of $\Omega_{\textit{t-exec}(\mathcal{P}_{H_1\lceil M_i})}$, for each finite prefix α' of α there is an element α_1 of $\textit{t-exec}(\Omega_{H_1\lceil M_i})$ such that $\alpha' \le \alpha_1$. It is enough to show that for each finite prefix α' of α there is also an element α_2 of $\textit{t-exec}(\Omega_{H_2\lceil M_i})$ such that $\alpha' \le \alpha_2$.

Let α' be a finite prefix of α such that there is an element α_1 of $\textit{t-exec}(\Omega_{H_1\lceil M_i})$ such that $\alpha' \le \alpha_1$. Thus, there is a time-enriched execution α'_1 of $\Omega_{H_1\lceil M_i}$ such that $\alpha' \le \textit{t-exec}(\alpha'_1)$. This means that there is a state q_1 of $H_1\lceil M_i$ such that $\alpha' \le \textit{t-exec}(q_1)$. From the definition of projection, there is a state q'_1 of H_1 such that $\alpha' \le \textit{t-exec}(q'_1\lceil M_i)$, and thus there is a timed execution α''_1 of $\textit{t-exec}(\Omega_{H_1})$ such that $\alpha' \le (\alpha''_1\lceil M_i)$. Consider a finite prefix α'''_1 of α''_1 such that $\alpha' \le (\alpha'''_1\lceil M_i)$. Then, $P_{\textit{t-exec}(\mathcal{P}_{H_1})}[C_{\alpha'''_1}] > 0$. Since $H_1 \equiv H_2$, $P_{\textit{t-exec}(\mathcal{P}_{H_2})}[C_{\alpha'''_1}] > 0$, which means that there is a timed execution α''_2 of $\Omega_{\textit{t-exec}(\mathcal{P}_{H_2})}$ such that $\alpha' \le (\alpha''_2\lceil M_i)$. Thus, there is a state q'_2 of H_2 such that $\alpha' \le \textit{t-exec}(q'_2\lceil M_i)$, and from the definition of projection, there is a state q_2 of $H_2\lceil M_i$ such that $\alpha' \le \textit{t-exec}(q_2)$. This implies that there is an element α'_2 of $\textit{t-exec}(\Omega_{H_2\lceil M_i})$ such that $\alpha' \le \alpha'_2$, which is sufficient to conclude. ■

9.6 Discussion

To our knowledge, no general probabilistic models with dense time have been proposed except for the automata of Courcoubetis, Alur and Dill [ACD91a, ACD91b]. In our model no probability distributions over passage of time are allowed within a probabilistic timed automaton; time can elapse probabilistically only within a probabilistic timed execution, and the associated probability distributions can be only discrete. We have chosen to define the timed model with such a restriction so that all the theory for the untimed model carries over.

Further work should investigate on the extension of our model to non-discrete probability distributions. A starting point could be the study of restricted forms of non-discrete distributions as it is done by Courcoubetis, Alur and Dill in [ACD91a, ACD91b]. Useful ideas can come from the work on stochastic process algebras of Götz, Herzog and Rettelbach [GHR93], Hillston [Hil94], and Bernardo, Donatiello and Gorrieri [BDG94].

Chapter 10

Direct Verification: Time Complexity

Part of this chapter is based on joint work with Anna Pogosyants and Isaac Saias; some of the ideas have been influenced by discussion with Lenore Zuck. The verification of the randomized dining philosophers algorithm of Lehmann and Rabin (Section 10.6) is based on joint work with Nancy Lynch and Isaac Saias [LSS94]; the verification of the randomized algorithm for agreement of Ben-Or (Section 10.8) is joint work with Anna Pogosyants and is a formalization of a proof that appears in the book on distributed algorithms of Nancy Lynch [Lyn95]. Close interaction with Anna Pogosyants lead us to the idea of the abstract complexity measures of Section 10.7.

10.1 General Considerations About Time

The direct analysis of a probabilistic timed automaton is carried out exactly in the same way as for untimed probabilistic automata. Thus, probabilistic statements and progress statements can be generalized directly, and the coin lemmas can be applied without any modification.

In this chapter we concentrate more on topics that are specific to the presence of time. In particular, it is now possible to enrich the notation for progress statements and verify some of the real-time properties of a probabilistic timed automaton. We extend the progress statements of Chapter 5 by adding a time parameter t: the expression $U \xrightarrow[p]{t} U'$ means that, starting from a state of U, a state of U' is reached within time t with probability at least p. Based on the new *timed progress statements* we show how to derive upper bounds on the worst expected time for progress.

We generalize the method for time complexity analysis to more abstract complexity measures. Then, rather than studying the expected time for progress, we study the expected abstract complexity for progress. We use abstract complexity to derive an upper bound on the worst expected time for decision of the randomized algorithm for agreement of Ben-Or that we presented in Chapter 5. Specifically, we show that under some conditions on the scheduling policy, each non-faulty process completes its i^{th} stage within some upper bound, and we show an upper bound on the expected number of stages that are necessary to reach agreement. In this case the abstract complexity is the number of stages. A direct analysis of the expected time for success in Ben-Or's algorithm would not be as easy since there is no useful upper bound on the time it takes to a process to move from a stage to the next stage.

Sections 10.2, 10.3, and 10.4 simply extend the definitions of Chapter 5 to the timed case; Section 10.5 shows how to derive upper bounds on the worst expected time for progress given a timed progress statement, and Section 10.7 shows how to derive upper bounds on the worst expected abstract complexity for progress given a timed progress statement with abstract complexity; Sections 10.6 and 10.8 present examples of application by proving that the randomized dining philosophers algorithm of Lehmann and Rabin guarantees progress in expected constant time and that the randomized agreement algorithm of Ben-Or guarantees agreement in expected exponential time.

10.2 Adversaries

An *adversary* for a probabilistic timed automaton M is a function $\mathcal{A}$ that takes a finite timed execution fragment α of M and returns a timed transition of M that leaves from $\mathit{lstate}(\alpha)$. Formally,

$$\mathcal{A} : \textit{t-frag}^*(M) \rightarrow \textit{t-trans}(M)$$

such that if $\mathcal{A}(\alpha) = (s, \mathcal{P})$, then $s = \mathit{lstate}(\alpha)$. Moreover, an adversary satisfies the following *consistency* condition: if $\mathcal{A}(\alpha) = (s, \mathcal{P})$, then for each prefix α' of some element α'' of Ω, $\mathcal{A}(\alpha \frown \alpha') = (\mathit{lstate}(\alpha'), \mathcal{P} \triangleright \alpha')$. Informally, consistency says that an adversary does not change its mind during a timed transition.

An adversary is *deterministic* if it returns either deterministic timed transitions of M or pairs of the form $(s, \mathcal{D}(s\delta))$, i.e., the next timed transition is chosen deterministically. Denote the set of adversaries and deterministic adversaries for a probabilistic timed automaton M by $\mathit{Advs}(M)$ and $\mathit{DAdvs}(M)$, respectively.

The definitions of an adversary schema and of the result of the interaction between an adversary and a probabilistic timed automaton is the same as for the untimed case (cf. Section 5.2), and thus we do not repeat them here.

To guarantee that our adversaries are well defined, we need to prove the following lemma.

Lemma 10.2.1 *If $(s, \mathcal{P})$ is a timed transition of a probabilistic timed automaton M, then for each prefix α' of some element α'' of Ω, $(\mathit{lstate}(\alpha'), \mathcal{P} \triangleright \alpha')$ is a timed transition of M.*

Proof. This is proved already in Proposition 9.3.5. ■

10.3 Event Schemas

As for the untimed case we need a mechanism to associate an event with each probabilistic timed execution fragment of a probabilistic timed automaton. Thus, an *event schema* is a function e that associates an event of the space $\mathcal{P}_H$ with each probabilistic timed execution fragment H of M. The notion of finite satisfiability extends directly from the untimed case. Observe that, although in $\mathcal{P}_H$ there can be uncountably many cones, each finitely satisfiable event can be expressed as the union of countably many disjoint cones. Furthermore, every uncountable family of cones contains at least two cones that are not disjoint.

The definition of a timed probabilistic statement extends directly from the untimed case, and similarly the definition of the concatenation of two event schemas extends directly. Therefore, we omit the definitions, which are identical to those of Chapter 5.

Proposition 10.3.1 *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 timed execution fragment H. From Proposition 9.3.3 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 10.3.2 $P_H[e_1 \circ_{Cones} e_2(H)] = \sum_{q \in Cones(H)} P_H[C_q] P_{H|q}[e_2(H|q)]$.

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

$$P_H[e_1 \circ_{Cones} e_2(H)] = \sum_{q \in Cones(H)} P_H[e_2(H|q)]. \tag{10.1}$$

From Proposition 9.3.3, for each $q \in Cones(H)$

$$P_H[e_2(H|q)] = P_H[C_q] P_{H|q}[e_2(H|q)]. \tag{10.2}$$

By substituting (10.2) in (10.1) we obtain the desired result. ■

Now it is possible to prove a concatenation property similar to the one for the untimed case.

Proposition 10.3.3 *Consider a probabilistic timed 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 timed execution fragment $q \in \Theta$. Let $H = prexec(M, \mathcal{A}, q)$. From Proposition 10.3.2,

$$P_H[e_1 \circ_{Cones} e_2(H)] = \sum_{q' \in Cones(H)} P_H[C_{q'}] P_{H|q'}[e_2(H|q')]. \tag{10.3}$$

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

$$H|q' = prexec(M, \mathcal{A}, q'), \tag{10.4}$$

where we use consistency for the base case. Thus, from our second hypothesis,

$$P_{H|q'}[e_2(H|q')]\ \mathcal{R}\ p_2. \tag{10.5}$$

By substituting (10.5) in (10.3), we obtain

$$P_H[e_1 \circ_{Cones} e_2(H)]\ \mathcal{R}\ p_2 \sum_{q' \in Cones(e_1(H))} P_H[C_{q'}]. \tag{10.6}$$

By using the fact that $Cones(H)$ is a characterization of $e_1(H)$ as a disjoint union of cones, Equation (10.6) can be rewritten into

$$P_H[e_1 \circ_{Cones} e_2(H)] \mathcal{R} p_2 P_H[e_1(H)]. \tag{10.7}$$

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

$$P_H[e_1 \circ_{Cones} e_2(H)] \mathcal{R} p_1 p_2. \tag{10.8}$$

This completes the proof. ■

10.4 Timed Progress Statements

As a special case of a probabilistic statement for the timed case we can add some features to the notation $X \xrightarrow[p]{}_{Advs} X'$. In particular we define a *timed progress statement* to assert that starting from a set of states U some other state of a set U' is reached within time t with probability at least p. Such a statement, which we denote by $U \xrightarrow[p]{t}_{Advs} U'$, or by $U \xrightarrow[p]{t} U'$ if *Advs* is clear from the context, is expressed by the probabilistic statement $\Pr_{Advs,U}(e_{U',t}) \geq p$, where the event schema $e_{U',t}$ applied to a timed probabilistic execution fragment H returns the set of timed executions α of Ω_H where a state from U' is reached within time t in $\alpha \triangleright q_0^H$. Such a set can be expressed as a union of cones, and therefore it is an event.

Similarly, the progress statements involving actions can be generalized to the timed framework. Thus, $V \xrightarrow[p]{t}_{Advs} V'$ is the probabilistic statement $\Pr_{Advs,\Theta_{V,V'}}(e_{V',t}) \geq p$, where $\Theta_{V,V'}$ is the set of finite timed 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, and the event schema $e_{V',t}$ applied to a timed probabilistic execution fragment H returns the set of timed executions α of Ω_H such that an action from V occurs in $\alpha \triangleright q_0^H$ within time t.

In order to generalize the concatenation theorem for progress statements, we need to extend the definition of a finite-history-insensitive adversary schema. Thus, an adversary schema *Advs* is *finite-history-insensitive* iff for each adversary $\mathcal{A}$ of *Advs* and each finite timed execution fragment α of M there is an adversary $\mathcal{A}'$ of *Advs* such that for each timed execution fragment α' such that $\alpha \leq \alpha'$, $\mathcal{A}(\alpha') = \mathcal{A}'(\alpha' \triangleright \alpha)$. Then, the following theorem is shown in the same way as for the untimed case.

Theorem 10.4.1 *Let Advs be finite-history-insensitive. If* $X \xrightarrow[p_1]{t_1}_{Advs} X'$ *and* $X' \xrightarrow[p_2]{t_2}_{Advs} X''$, *then* $X \xrightarrow[p_1 p_2]{t_1+t_2}_{Advs} X''$. ■

10.5 Time Complexity

In this section we show how to study the time complexity of a randomized distributed algorithm. We start by defining how to compute a worst expected time, and then we show how it is possible to derive upper bounds on the worst expected running time of an algorithm based on timed progress statements.

10.5.1 Expected Time of Success

Let e be a finitely satisfiable event schema and suppose that $P_H[e(H)] = 1$, i.e., that the property described by e is satisfied in H with probability 1. Let $Cones(H)$ be a characterization of $e(H)$ as a disjoint union of cones, where each element of $Cones(H)$ identifies the first point along a timed execution where the property denoted by e is satisfied. Then, we can compute the expected time to satisfy the property identified by e as

$$\sum_{q \in Cones(H)} P_H[C_q](ltime(q \rhd q_0^H)). \tag{10.9}$$

In general, if e is a finitely satisfiable event-schema and $Cones(H)$ identifies the first point along a timed execution where the property identified by e is satisfied, then for each probabilistic timed execution fragment H of M we define $E_H[e]$, the expected time to satisfy e in H, as follows.

$$E_H[e] = \begin{cases} \sum_{q \in Cones(H)} P_H[C_q](ltime(q \rhd q_0^H)) & \text{if } P_H[e(H)] = 1 \\ \infty & \text{otherwise.} \end{cases} \tag{10.10}$$

Then, the question is the following: are there easy ways to compute upper bounds on the expected time for success in a randomized algorithm without computing explicitly (10.10)? We give a positive answer to this question.

10.5.2 From Timed Progress Statements to Expected Times

Timed progress statements can be used to analyze the time complexity of a randomized algorithm. The main idea for the analysis is expressed by Proposition 10.5.1. Suppose that we know the following:

$$\begin{cases} U \xrightarrow[p]{t}_{Advs} U' \\ U \Rightarrow (U \; Unless \; U'). \end{cases} \tag{10.11}$$

Then, if $Advs$ is finite-history-insensitive and $s\delta \notin \Omega_{\mathcal{A}(s)}$ for each $\mathcal{A} \in Advs$ and each $s \in U$, we know from Proposition 5.5.6 that $U \xrightarrow[1]{}_{Advs} U'$. Let e be a finitely satisfiable event schema, and let $Cones$ express the points of satisfaction of e. Suppose that for each probabilistic timed execution fragment H and each state q of H, if there is no prefix q' of q such that $q' \in Cones(H)$, then $e(H \rhd q) = e(H) \rhd q$ and $Cones(H \rhd q) = Cones(H) \rhd q$ (e.g., e can express the property of reaching some state in a set U'', or the property of performing some action). Let

$$E_{U,Advs}[e] \;\stackrel{\triangle}{=}\; sup_{s \in U, \mathcal{A} \in Advs} E_{prexec(M,\mathcal{A},s)}[e]. \tag{10.12}$$

Then the following property is valid.

Proposition 10.5.1

$$E_{U,Advs}[e] \leq t + pE_{U',Advs}[e] + (1-p)E_{U,Advs}[e]. \tag{10.13}$$

Proof. We prove (10.13) by distinguishing four cases.

1. $E_{U',Advs}[e] \geq E_{U,Advs}[e]$.

   In this case (10.13) is satisfied trivially.

2. $E_{U,Advs}[e] = \infty$ and $p < 1$.

   Also in this case (10.13) is satisfied trivially.

3. $E_{U,Advs}[e] = \infty$ and $p = 1$.

   We show that $E_{U',Advs}[e] = \infty$, which is enough to satisfy (10.13). Suppose by contradiction that $E_{U',Advs}[e] < \infty$. Then we distinguish the following cases.

   (a) There is an adversary $\mathcal{A}$ of *Advs* and a state s of U such that $P_{prexec(M,\mathcal{A},s)}[e(prexec(M,\mathcal{A},s))] < 1$.

   (b) It is not the case that there is an adversary $\mathcal{A}$ of *Advs* and a state s of U such that $P_{prexec(M,\mathcal{A},s)}[e(prexec(M,\mathcal{A},s))] < 1$.

   For Case (a), let $Cones_{U'}$ be the function that expresses the points of satisfaction of $e_{U'}$, and let H be $prexec(M,\mathcal{A},s)$, where $P_{prexec(M,\mathcal{A},s)}[e(prexec(M,\mathcal{A},s))] < 1$. Then,

$$P_H[e(H)] \geq \sum_{q \in Cones_{U'}(H)} P_H[C_q] P_{H \triangleright q}(e(H \triangleright q)), \tag{10.14}$$

   i.e., the probability of satisfying e is not smaller than the probability of reaching U' and then from there satisfying e. From the finite-history-insensitivity of *Advs*, for each state q of $Cones_{U'}(H)$ there is an adversary $\mathcal{A}'$ of *Advs* such that $H \triangleright q = prexec(M, \mathcal{A}', lstate(q))$, and thus, since $E_{U',Advs}[e] < \infty$, $P_{H \triangleright q}(e(H \triangleright q)) = 1$. By substituting this result in (10.14), we get

$$P_H[e(H)] \geq \sum_{q \in Cones_{U'}(H)} P_H[C_q]. \tag{10.15}$$

   Since $p = 1$, the right side of (10.15) is equal to 1, i.e., $P_H[e(H)] \geq 1$, a contradiction.

   For Case (b), let $Cones_{U'}$ be a function that expresses the points of satisfaction of e_U, and, for each $d > 0$, let $Cones_d$ be a function that expresses the event of reaching time d as a union of disjoint cones. From the definition of a probabilistic timed execution, we know that $Cones_d$ exists and that for each probabilistic timed execution fragment H and each $q \in Cones_d(H)$, $ltime(q \triangleright q_0^H) = d$. Let H be $prexec(M,\mathcal{A},s)$. From (10.10) the expected time for success for e is

$$E_H[e] = \sum_{q \in Cones(H)} P_H[C_q] ltime(q \triangleright q_0^H). \tag{10.16}$$

   Let ϵ be an arbitrary positive number. Let Θ_1 be the set of elements q of $Cones_{U'}(H)$ such that $ltime(q \triangleright q_0^H) < t + \epsilon$, and let H_2 be the set of elements q of $Cones_{t+\epsilon}(H)$ that do not have any prefix in Θ_1. Since $P_H[e_U(H)] = 1$, then $P_H[\cup_{q \in \Theta_1 \cup \Theta_2} C_q] = 1$. Moreover, by hypothesis, $P_H[\cup_{q \in Cones(H)} C_q] = 1$. Thus, observe that each element of $Cones(H)$ has either a proper prefix or a suffix in $\Theta_1 \cup \Theta_2$. In fact, if there is an element q of $Cones(H)$ that has no prefix nor suffix in $\Theta_1 \cup \Theta_2$, then the cone C_q would not be part of $\cup_{q \in \Theta_1 \cup \Theta_2} C_q$, contradicting the hypothesis that $P_H[\cup_{q \in Cones(H)} C_q] = 1$. Similarly, we can show that for each element q of $\Theta_1 \cup \Theta_2$ has either a prefix or a proper suffix in $Cones(H)$. Thus, $Cones(H)$ can be partitioned into two sets Θ^p and Θ^s of elements that have a proper prefix and a suffix, respectively, in $\Theta_1 \cup \Theta_2$, and $\Theta_1 \cup \Theta_2$ can be partitioned into two sets $\Theta^p_{1,2}$ and $\Theta^s_{1,2}$ of elements that have a prefix and a proper suffix, respectively, in $Cones(H)$. Based on these observations, the right side of Equation(10.16) can be rewritten into

$$\begin{aligned}&\left(\sum_{q\in\Theta^p}\;\sum_{q'\in\Theta^s_{1,2}|q'\leq q} P_H[C_{q'}]P_{H\triangleright q'}[C_{q\triangleright q'}](ltime(q'\triangleright q_0^H)+ltime(q\triangleright q'))\right) \\ &+\left(\sum_{q\in\Theta^s}\;\sum_{q'\in\Theta^p_{1,2}|q\leq q'} P_H[C_q]P_{H\triangleright q}[C_{q'\triangleright q}]ltime(q\triangleright q_0^H)\right).\end{aligned} \tag{10.17}$$

Observe that for each $q \in \Theta^s$, $\sum_{q'\in\Theta^p_{1,2}|q\leq q'} P_{H\triangleright q}[C_{q'\triangleright q}] = 1$, and observe that for each $q' \in \Theta^s_{1,2}$, $\sum_{q\in\Theta^p|q'\leq q} P_{H\triangleright q'}[C_{q\triangleright q'}] = 1$. By exchanging the sums in (10.17) and using some simple algebraic manipulations, we obtain

$$\begin{aligned}&\left(\sum_{q'\in\Theta^s_{1,2}} P_H[C_{q'}]\left(ltime(q'\triangleright q_0^H)+\sum_{q\in\Theta^p|q'\leq q} P_{H\triangleright q'}[C_{q\triangleright q'}]ltime(q\triangleright q')\right)\right) \\ &+\left(\sum_{q'\in\Theta^p_{1,2}}\;\sum_{q\in\Theta^s|q\leq q'} P_H[C_q]P_{H\triangleright q}[C_{q'\triangleright q}]ltime(q\triangleright q_0^H)\right).\end{aligned} \tag{10.18}$$

In the first summand, since from the properties of e for each $q' \in \Theta^s_{1,2}$, $e(H\triangleright q') = e(H)\triangleright q'$, the subexpression $\sum_{q\in\Theta^p|q'\leq q} ltime(q\triangleright q')P_{H\triangleright q'}[C_{q\triangleright q'}]$ denotes $E_{H\triangleright q'}[e]$. In the second summand, observe that for each $q' \in \Theta^p_{1,2}$ there is exactly one element q of Θ^s such that $q \leq q'$. Moreover, $P_H[C_q]P_{H\triangleright q}[C_{q'\triangleright q}] = P_H[C_{q'}]$. Thus, from (10.18) we obtain

$$\begin{aligned}E_H[e] \quad &\leq \quad \left(\sum_{q'\in\Theta^s_{1,2}} P_H[C_{q'}](ltime(q'\triangleright q_0^H)+E_{H\triangleright q'}[e])\right) \\ &+ \quad \left(\sum_{q'\in\Theta^p_{1,2}} P_H[C_{q'}]ltime(q'\triangleright q_0^H)\right).\end{aligned} \tag{10.19}$$

By repartitioning $\Theta^s_{1,2} \cup \Theta^p_{1,2}$ into Θ_1 and Θ_2, and by observing that for each element q of Θ_1 $ltime(q\triangleright q_0^H) < t+\epsilon$, and for each element q of Θ_2 $ltime(q\triangleright q_0^H) = t+\epsilon$, (10.19) can be rewritten into

$$\begin{aligned}E_H[e] \leq (t+\epsilon)&\left(\sum_{q\in\Theta^s_{1,2}\cap\Theta_1} P_H[C_q]E_{H\triangleright q}[e])\right)+\left(\sum_{q\in\Theta^p_{1,2}\cap\Theta_1} P_H[C_q]E_{H\triangleright q}[e]\right) \\ +&\left(\sum_{q\in\Theta^s_{1,2}\cap\Theta_2} P_H[C_q]E_{H\triangleright q}[e]\right)+\left(\sum_{q\in\Theta^p_{1,2}\cap\Theta_2} P_H[C_q]E_{U,Advs}[e]\right),\end{aligned} \tag{10.20}$$

where we have added $E_{H \rhd q}[e]$ in the upper right summand and $E_{U,Advs}[e]$ in the lower right summand. Since *Advs* is finite history insensitive, for each $q \in \Theta_1 \cup \Theta_2$ there is an adversary $\mathcal{A}'$ of *Advs* such that $(H \rhd q) = prexec(M, \mathcal{A}, lstate(q))$. Thus, (10.20) can be rewritten into

$$E_H[e] \le (t+\epsilon)\left(\sum_{q \in \Theta_1} P_H[C_q] E_{U',Advs}[e])\right) + \left(\sum_{q \in \Theta_2} P_H[C_q] E_{U,Advs}[e]\right), \tag{10.21}$$

where we have used $U \Rightarrow (U\ Unless\ U')$ to say that the last states of the elements of Θ_2 are in U. Observe that $\sum_{q \in \Theta_1} P_H[C_q]$ is $P_H[e_{U',t}(H)]$, which is 1 by hypothesis. Since by hypothesis $E_{U',Advs}[e] < \infty$, from (10.21) we derive that $E_{U,Advs}[e] < \infty$, a contradiction.

4. $E_{U,Advs}[e] < \infty$, $E_{U',Advs}[e] < \infty$, and $E_{U',Advs}[e] \le E_{U',Advs}[e]$.

   Let $\mathcal{A}$ be an adversary of *Advs* and s be a state of U. Let H be $prexec(M, \mathcal{A}, s)$. Let ϵ be any positive real number. Equation (10.21) can be derived also in this case using the same identical argument as before. Since we have assumed that $E_{U',Advs}[e] \le E_{U,Advs}[e]$, the lowest possible value of the right side of (10.21) occurs by giving U' the lowest possible probability, which is p. Thus, (10.21) becomes

   $$E_H[e] \le (t+\epsilon) p E_{U',Advs}[e] + (1-p) E_{U,Advs}[e]. \tag{10.22}$$

   Since Equation (10.22) is valid for any adversary *Advs* and any state of U, we obtain timed execution fragment

   $$E_{U,Advs}[e] \le (t+\epsilon) p E_{U',Advs}[e] + (1-p) E_{U,Advs}[e]. \tag{10.23}$$

   Since Equation (10.23) is valid for every ϵ, Equation (10.23) is valid also for the infimum of the values that ϵ can have, i.e., 0, and thus,

   $$E_{U,Advs}[e] \le t + p E_{U',Advs}[e] + (1-p) E_{U,Advs}[e]. \tag{10.24}$$

   This completes the proof. ■

Example 10.5.1 (From timed progress to expected time) As a simple example of application of Proposition 10.5.1, suppose that e expresses the property of reaching U'. Then, we know by definition that $E_{U',Advs}[e] = 0$. By applying Equation (10.13), we obtain $E_{U,Advs}[e] \le t + (1-p) E_{U,Advs}[e]$, which gives $E_{U,Advs}[e] \le t/p$, i.e., the expected time to reach U' from U is at most t/p. Informally speaking, we can view the process of reaching U' as a sequence of Bernoulli trials, each one performed every t time units. At time t, with probability p we have reached U', and with probability $(1-p)$ we are still in U, and thus we apply the same experiment again. The expected number of rounds of such a process is $1/p$, and thus the expected time for success is t/p. Suppose now that we know the following,

$$\left\{ \begin{array}{ll} U_0 \xrightarrow[p_1]{t_1}_{Advs} U_1 & U_0 \Rightarrow (U_0\ Unless\ U_1) \\ U_1 \xrightarrow[p_2]{t_2}_{Advs} U_2 & U_1 \Rightarrow (U_1\ Unless\ U_2), \end{array} \right. \tag{10.25}$$

and suppose that e expresses the property of reaching U_2. Then, we know that $E_{U_2,Advs}[e] = 0$. By applying Proposition 10.5.1, we obtain

$$\left\{\begin{array}{l} E_{U_0,Advs}[e] \leq t_1 + p_1 E_{U_1,Advs}[e] + (1-p_1)E_{U_0,Advs}[e] \\ E_{U_1,Advs}[e] \leq t_2 + (1-p_2)E_{U_1,Advs}[e]. \end{array}\right. \tag{10.26}$$

From simple algebraic manipulations (10.26) becomes

$$\left\{\begin{array}{l} E_{U_0,Advs}[e] \leq t_1/p_1 + E_{U_1,Advs}[e] \\ E_{U_1,Advs}[e] \leq t_2/p_2, \end{array}\right. \tag{10.27}$$

and thus, after substituting the second inequality in the first inequality,

$$\left\{\begin{array}{l} E_{U_0,Advs}[e] \leq t_1/p_1 + t_2/p_2 \\ E_{U_1,Advs}[e] \leq t_2/p_2. \end{array}\right. \tag{10.28}$$

Suppose now that in addition to (10.25) we know that

$$\left\{\begin{array}{l} U_0 \xrightarrow[p_3]{t_3}_{Advs} U_2 \\ U_0 \Rightarrow (U_0 \; Unless \; U_2), \end{array}\right. \tag{10.29}$$

which is possible if $U_1 \subseteq U_0 \cup U_2$. Then, from Proposition 10.5.1 we get

$$E_{U_0,Advs}[e] \leq t_3/p_3, \tag{10.30}$$

which added to (10.28) gives

$$\left\{\begin{array}{l} E_{U_0,Advs}[e] \leq min(t_1/p_1 + t_2/p_2, t_3/p_3) \\ E_{U_1,Advs}[e] \leq t_2/p_2. \end{array}\right. \tag{10.31}$$

Therefore, more information may give us the possibility to prove better bounds. ■

Proposition 10.5.1 can be proved also for timed progress statements that involve sets of actions rather than sets of states. Let V, V' denote two sets of actions, and let $Advs$ be an adversary schema. Suppose that

$$V \xrightarrow[p]{t}_{Advs} V'. \tag{10.32}$$

Let e be a finitely satisfiable event schema, and let $Cones$ express the points of satisfaction of e. Suppose that for each probabilistic timed execution fragment H and each state q of H, if there is no prefix q' of q such that $q' \in Cones(H)$, then $e(H \triangleright q) = e(H) \triangleright q$ and $Cones(H \triangleright q) = Cones(H) \triangleright q$. Let $E_{V,V',Advs}[e]$ denote $sup_{q \in \Theta_{V,V'}, \mathcal{A} \in Advs} E_{prexec(M,\mathcal{A},q)}[e]$. Let $\Theta_{V'}$ denote the set of finite execution fragments of M whose last action is in V', and let $E_{V',Advs}[e]$ denote $sup_{q \in \Theta_{V'}, \mathcal{A} \in Advs} E_{prexec(M,\mathcal{A},q)}[e]$. Suppose that $q'\delta \notin \Omega_{\mathcal{A}(q)}$ for each q', each $\mathcal{A} \in Advs$ and each $q \in \Theta_{V,V'}$. Then the following proposition is valid.

Proposition 10.5.2

1. *$E_{V,V',Advs}[e] \leq t + pE_{V',Advs}[e] + (1-p)E_{V,V',Advs}[e]$, and*

2. *for each set of actions V'', $E_{V',Advs}[e] \leq E_{V',V'',Advs}[e]$.*

Proof. The proof of the first item follows the lines of the proof of Proposition 10.5.1; the proof of the second item follows from the fact that $\Theta_{V'} \subseteq \Theta_{V',V''}$. ■

10.6 Example: Randomized Dining Philosophers

To illustrate the use of timed progress statements for the analysis of an algorithm, we reconsider the randomized dining philosophers algorithm of Lehmann and Rabin, and we show that, under the condition that each process has a minimum speed, progress is guaranteed within expected constant time. First, we show how to add time to the probabilistic automaton that describes the algorithm; then, we add time limitations to the progress statements that we used in Section 6.3.3 and we derive the upper bound on the expected time for progress; finally we repeat the low level proof observing that the coin lemmas are applied in the same way as for the untimed case.

10.6.1 Representation of the Algorithm

The probabilistic timed automaton that represent the Algorithm of Lehmann and Rabin can be obtained directly from the probabilistic automaton of Section 6.3.2 by adding arbitrary self-loop time-passage transition from each state (same as the patient construction of Example 9.2.1). Then, in order to enforce a lower bound on the speed of each process, we impose some limitations on the adversaries that act on M. For convenience, but without loss of generality, we assume that from any point each process in its trying or exit region performs one transition within time 1. Thus, the adversary schema that we use on M is the set of adversaries $\mathcal{A}$ for M such that for each finite timed execution fragment α of M,

1. $P_{prexec(M,\mathcal{A},\alpha)}[\mathit{frag}^{\infty}(M)] = 1$, and
2. for each element α' of $\Omega_{prexec(M,\mathcal{A},\alpha)}$ there is no pair of prefixes $\alpha_1 \leq \alpha_2$ of $\alpha' \triangleright \alpha$ and no process i such that process i is in its trying or exit region in $\mathit{lstate}(\alpha_1)$, $\mathit{ltime}(\alpha_2 \triangleright \alpha_1) > 1$, and process i does not perform any discrete transition in $\alpha_2 \triangleright \alpha_1$.

We call this adversary schema *Unit-Time*.

Remark 10.6.1 Observe that in Condition 1 we require the probability of the admissible executions to be 1 rather than requiring the sample space to contain only admissible executions. The reason for using probabilities is technical and is due to the fact that the sample space of a probabilistic timed executions always contains Zeno timed executions, even though they occur with probability 0. From the practical point of view all the Zeno timed executions can be ignored.

In other words, it is not necessary to know the intricacies of the definition of a probabilistic timed executions since they are used only to guarantee that the events of interest are measurable. From the point of view of verifying the correctness of a randomized distributed algorithm, as long as Zeno timed executions occur only with probability 0, it is possible to think that Zeno timed executions do not occur at all. ■

Remark 10.6.2 (Alternative approach) Another alternative approach to modeling the algorithm of Lehmann and Rabin, which we do not use here, is to augment the probabilistic automaton of Section 6.3.2 with an upper bound for each process i to the time by which process i must perform a transition, and to allow a time-passage transition only when no process goes beyond its upper bound. Of course the upper bounds need to be updated opportunely within a transition. In this case the condition imposed on an adversary would be just that time advances unboundedly with probability 1. ■

10.6.2 The High Level Proof

The high level proof consists of the same progress statements that we used in Section 6.3.3 together with a time bound. Specifically, we use the following timed progress statements.

$$\begin{array}{ll}
\mathcal{T} \xrightarrow[1]{2} \mathcal{RT} \cup \mathcal{C} & \text{(Proposition 10.6.3)}, \\
\mathcal{RT} \xrightarrow[1]{3} \mathcal{F} \cup \mathcal{G} \cup \mathcal{P} & \text{(Proposition 10.6.15)}, \\
\mathcal{F} \xrightarrow[1/2]{2} \mathcal{G} \cup \mathcal{P} & \text{(Proposition 10.6.14)}, \\
\mathcal{G} \xrightarrow[1/4]{5} \mathcal{P} & \text{(Proposition 10.6.11)}, \\
\mathcal{P} \xrightarrow[1]{1} \mathcal{C} & \text{(Proposition 10.6.1)}.
\end{array}$$

By combining the statements above by means of Proposition 5.5.3 and Theorem 10.4.1 we obtain

$$\mathcal{T} \xrightarrow[1/8]{13} \mathcal{C}. \tag{10.33}$$

Observing that if some process is in the trying region then some process is in the trying region unless some process gets to the critical region, we apply Proposition 10.5.1 and we obtain that the expected time to reach $\mathcal{C}$ from $\mathcal{RT}$ is at most 104, i.e., the algorithm of Lehmann and Rabin guarantees progress within expected constant time.

10.6.3 The Low Level Proof

We now prove the timed progress statements of Section 10.6.2. The proofs are exactly the same as the proofs given in Section 6.3.4 with the difference that in this case we consider also time bounds and we consider only admissible timed execution fragments since we know that they occur with probability 1.

Proposition 10.6.1 *If some process is in P, then some process enters C within time 1, i.e.,*

$$\mathcal{P} \xrightarrow[1]{1} \mathcal{C}.$$

Proof. Let i be the process in P. Then, from the definition of *Unit-Time*, process i is scheduled within time 1, and enters C. ■

Lemma 10.6.2 *If some process is in its Exit region, then it will enter R within time* 3.

Proof. The process needs to perform two transitions to relinquish its two resources, and then one transition to send a `rem` message to the user. Every adversary of *Unit-Time* guarantees that those three transitions are performed within time 3. ■

Proposition 10.6.3 $\mathcal{T} \xrightarrow{2} \mathcal{RT} \cup \mathcal{C}$.

Proof. From Lemma 6.3.2, every process that begins in E_F or E_S relinquishes its resources within time 2 . If no process begins in C or enters C in the meantime, then the state reached at this point is a state of $\mathcal{RT}$; otherwise, the starting state or the state reached when the first process enters C is a state of $\mathcal{C}$. ■

We now turn to the proof of $\mathcal{G} \xrightarrow[1/4]{5} \mathcal{P}$. The following lemmas form a detailed cases analysis of the different situations that can arise in states of $\mathcal{G}$. Informally, each lemma shows that a specific coin event is a sub-event of the properties of reaching some other state. Here we do not repeat the proof of Lemma 6.3.4 since it does not depend on timing issues.

Lemma 10.6.4

1. *Let* $X_{i-1} \in \{E_R, R, F\}$ *and* $X_i = \underset{\leftarrow}{W}$. *If* $\mathit{FIRST}(\texttt{flip}_{i-1}, \texttt{left})$, *then, within time* 1, *either* $X_{i-1} = P$ *or* $X_i = S$.

2. *Let* $X_{i-1} = D$ *and* $X_i = \underset{\leftarrow}{W}$. *If* $\mathit{FIRST}(\texttt{flip}_{i-1}, \texttt{left})$, *then, within time* 2, *either* $X_{i-1} = P$ *or* $X_i = S$.

3. *Let* $X_{i-1} = S$ *and* $X_i = \underset{\leftarrow}{W}$. *If* $\mathit{FIRST}(\texttt{flip}_{i-1}, \texttt{left})$, *then, within time* 3, *either* $X_{i-1} = P$ *or* $X_i = S$.

4. *Let* $X_{i-1} = W$ *and* $X_i = \underset{\leftarrow}{W}$. *If* $\mathit{FIRST}(\texttt{flip}_{i-1}, \texttt{left})$, *then, within time* 4, *either* $X_{i-1} = P$ *or* $X_i = S$.

Proof. The four proofs start in the same way. Let s be a state of M satisfying the respective properties of items *1* or *2* or *3* or *4*. Let $\mathcal{A}$ be an adversary of *Unit-Time*, and let α be an admissible timed execution of $\Omega_{prexec(M,\{s\},\mathcal{A})}$ where the result of the first coin flip of process $i-1$, if it occurs, is `left`.

1. By hypothesis and Lemma 6.3.4, $i-1$ does not hold any resource at the beginning of α and has to obtain Res_{i-2} (its left resource) before pursuing Res_{i-1}. From the definition of *Unit-Time*, i performs a transition within time 1 in α. If $i-1$ does not hold Res_{i-1} when i performs this transition, then i progresses into configuration S. If not, it must be the case that $i-1$ succeeded in getting it in the meanwhile. But, in this case, since $i-1$ flips `left`, Res_{i-1} was the second resource needed by $i-1$ and $i-1$ therefore entered P.

2. If $X_i = S$ within time 1, then we are done. Otherwise, process $i-1$ performs a transition within time 1. Let $\alpha = \alpha_1 \frown \alpha_2$ such that the last transition of α_1 is the first transition taken by process $i-1$. Then $X_{i-1}(\mathit{fstate}(\alpha_2)) = F$ and $X_i(\mathit{fstate}(\alpha_2)) = \underset{\leftarrow}{W}$. Since process $i-1$ did not flip any coin during α_1, from the finite-history-insensitivity of *Unit-Time* and Item *1* we conclude.

3. If $X_i = S$ within time 1, then we are done. Otherwise, process $i-1$ performs a transition within time 1. Let $\alpha = \alpha_1 \frown \alpha_2$ such that the last transition of α_1 is the first transition taken by process $i-1$. If $X_{i-1}(\mathit{fstate}(\alpha_2)) = P$ then we are also done. Otherwise it must be the case that $X_{i-1}(\mathit{fstate}(\alpha_2)) = D$ and $X_i(\mathit{fstate}(\alpha_2)) = \underset{\leftarrow}{W}$. Since process $i-1$ did not flip any coin during α_1, from the finite-history-insensitivity of *Unit-Time* and Item *2* we conclude.

4. If $X_i = S$ within time 1, then we are done. Otherwise, process i checks its left resource within time 1 and fails, process $i-1$ gets its right resource before, and hence reaches at least state S. Let $\alpha = \alpha_1 \frown \alpha_2$ where the last transition of α_1 is the first transition of α that leads process $i-1$ to state S. Then $X_{i-1}(\mathit{fstate}(\alpha_2)) = S$ and $X_i(\mathit{fstate}(\alpha_2)) = \underleftarrow{W}$. Since process $i-1$ did not flip any coin during α_1, from the finite-history-insensitivity of *Unit-Time* and Item *3* we conclude. ■

Lemma 10.6.5 *Assume that* $X_{i-1} \in \{E_R, R, T\}$ *and* $X_i = \underleftarrow{W}$*. If* $\mathit{FIRST}(\mathtt{flip}_{i-1}, \mathtt{left})$*, then, within time* 4*, either* $X_{i-1} = P$ *or* $X_i = S$*.*

Proof. Follows directly from Lemma 10.6.4 after observing that $X_{i-1} \in \{E_R, R, T\}$ is equivalent to $X_{i-1} \in \{E_R, R, F, W, S, D, P\}$. ■

The next lemma is a useful tool for the proofs of Lemmas 10.6.7, 10.6.8, and 10.6.9. It is just repeated from Section 6.3.4.

Lemma 10.6.6 *Let* $X_i \in \{\underleftarrow{W}, \underleftarrow{S}\}$ *or* $X_i \in \{E_R, R, F, \underleftarrow{D}\}$ *with* $\mathit{FIRST}(\mathtt{flip}_i, \mathtt{left})$*. Furthermore, let* $X_{i+1} \in \{\underrightarrow{W}, \underrightarrow{S}\}$ *or* $X_{i+1} \in \{E_R, R, F, \underrightarrow{D}\}$ *with* $\mathit{FIRST}(\mathtt{flip}_{i+1}, \mathtt{right})$*. Then the first of the two processes* i *or* $i+1$ *testing its second resource enters* P *after having performed this test (if this time ever comes).*

Proof. By Lemma 6.3.4 Res_i is free. Moreover, Res_i is the second resource needed by both i and $i+1$. Whichever tests for it first gets it and enters P. ■

Lemma 10.6.7 *If* $X_i = \underleftarrow{S}$ *and* $X_{i+1} \in \{\underrightarrow{W}, \underrightarrow{S}\}$ *then, within time* 1*, one of the two processes* i *or* $i+1$ *enters* P*. The same result holds if* $X_i \in \{\underleftarrow{W}, \underleftarrow{S}\}$ *and* $X_{i+1} = \underrightarrow{S}$*.*

Proof. Being in state S, process i tests its second resource within time 1. An application of Lemma 10.6.6 finishes the proof. ■

Lemma 10.6.8 *Let* $X_i = \underleftarrow{S}$ *and* $X_{i+1} \in \{E_R, R, F, \underrightarrow{D}\}$*. If* $\mathit{FIRST}(\mathtt{flip}_{i+1}, \mathtt{right})$*, then, within time* 1*, one of the two processes* i *or* $i+1$ *enters* P*. The same result holds if* $X_i \in \{E_R, R, F, D\}$*,* $X_{i+1} = \underrightarrow{S}$ *and* $\mathit{FIRST}(\mathtt{flip}_i, \mathtt{left})$*.*

Proof. Being in state S, process i tests its second resource within time 1. An application of Lemma 10.6.6 finishes the proof. ■

Lemma 10.6.9 *Assume that* $X_{i-1} \in \{E_R, R, T\}$*,* $X_i = \underleftarrow{W}$*, and* $X_{i+1} \in \{E_R, R, F, \underrightarrow{W}, \underrightarrow{D}\}$*. If* $\mathit{FIRST}(\mathtt{flip}_{i-1}, \mathtt{left})$ *and* $\mathit{FIRST}(\mathtt{flip}_{i+1}, \mathtt{right})$*, then, within time* 5*, one of the three processes* $i-1$*,* i *or* $i+1$ *enters* P*.*

Proof. Let s be a state of M such that $X_{i-1}(s) \in \{E_R, R, T\}$, $X_i(s) = \underleftarrow{W}$, and $X_{i+1}(s) \in \{E_R, R, F, \underrightarrow{W}, \underrightarrow{D}\}$. Let $\mathcal{A}$ be an adversary of *Unit-Time*, and let α be an admissible timed execution of $\Omega_{\mathit{prexec}(M,\{s\},\mathcal{A})}$ where the result of the first coin flip of process $i-1$ is `left` and the result of the first coin flip of process $i+1$ is `right`. By Lemma 10.6.5, within time 4 either process $i-1$ reaches configuration P in α or process i reaches configuration $\underleftarrow{S}$ in α. If $i-1$

reaches configuration P, then we are done. If not, then let $\alpha = \alpha_1 \frown \alpha_2$ such that $\mathit{lstate}(\alpha_1)$ is the first state s' of α with $X_i(s') = \underleftarrow{S}$. If $i+1$ enters P before the end of α_1, then we are done. Otherwise, $X_{i+1}(\mathit{fstate}(\alpha_2))$ is either in $\{\underrightarrow{W}, \underrightarrow{S}\}$ or it is in $\{E_R, R, F, \underrightarrow{D}\}$ and process $i+1$ has not flipped any coin yet in α. From the finite-history-insensitivity of *Unit-Time* we can then apply Lemma 10.6.6: within time 1 process i tests its second resource and by Lemma 10.6.6 process i enters P if process $i+1$ did not check its second resource in the meantime. If process $i+1$ checks its second resource before process i does the same, then by Lemma 10.6.6 process $i+1$ enters P. ■

Lemma 10.6.10 *Assume that $X_{i+2} \in \{E_R, R, T\}$, $X_{i+1} = \underrightarrow{W}$, and $X_i \in \{E_R, R, F, \underleftarrow{W}, \underleftarrow{D}\}$. If $\mathit{FIRST}(\mathtt{flip}_i, \mathtt{left})$ and $\mathit{FIRST}(\mathtt{flip}_{i+2}, \mathtt{right})$, then, within time 5, one of the three processes i, $i+1$ or $i+2$, enters P.*

Proof. The proof is analogous to the one of Lemma 10.6.9. This lemma is the symmetric case of Lemma 10.6.9. ■

Proposition 10.6.11 *Starting from a global configuration in $\mathcal{G}$, then, with probability at least 1/4, some process enters P within time 5. Equivalently:*

$$\mathcal{G} \xrightarrow[1/4]{5} \mathcal{P}.$$

Proof. Lemmas 10.6.7 and 10.6.8 jointly treat the case where $X_i = \underleftarrow{S}$ and $X_{i+1} \in \{E_R, R, F, \underrightarrow{\#}\}$ and the symmetric case where $X_i \in \{E_R, R, F, \underleftarrow{\#}\}$ and $X_{i+1} = \underrightarrow{S}$; Lemmas 10.6.9 and 10.6.10 jointly treat the case where $X_i = \underleftarrow{W}$ and $X_{i+1} \in \{E_R, R, F, \underrightarrow{W}, \underrightarrow{D}\}$ and the symmetric case where $X_i \in \{E_R, R, F, \underleftarrow{W}, \underleftarrow{D}\}$ and $X_{i+1} = \underrightarrow{W}$.

Specifically, each lemma shows that a compound event of the kind $\mathit{FIRST}(\mathtt{flip}_i, x)$ and $\mathit{FIRST}(\mathtt{flip}_j, y)$ leads to $\mathcal{P}$. Each of the basic events $\mathit{FIRST}(\mathtt{flip}_i, x)$ has probability at least 1/2. From Lemma 6.2.4 each of the compound events has probability at least 1/4. Thus the probability of reaching $\mathcal{P}$ within time 5 is at least 1/4. ■

We now turn to $\mathcal{F} \xrightarrow[1/2]{2} \mathcal{G} \cup \mathcal{P}$. The proof is divided in two parts and constitute the global argument of the proof of progress, i.e., the argument that focuses on the whole system rather than on a couple of processes.

Lemma 10.6.12 *Start with a state s of $\mathcal{F}$. If there exists a process i for which $X_i(s) = F$ and $(X_{i-1}, X_{i+1}) \neq (\underrightarrow{\#}, \underleftarrow{\#})$, then, with probability at least 1/2 a state of $\mathcal{G} \cup \mathcal{P}$ is reached within time 1.*

Proof. If $s \in \mathcal{G} \cup \mathcal{P}$, then the result is trivial. Let s be a state of $\mathcal{F} - (\mathcal{G} \cup \mathcal{P})$ and let i be such that $X_i(s) = F$ and $(X_{i-1}, X_{i+1}) \neq (\underrightarrow{\#}, \underleftarrow{\#})$. Assume without loss of generality that $X_{i+1} \neq \underleftarrow{\#}$, i.e., $X_{i+1} \in \{E_R, R, F, \underrightarrow{\#}\}$. The case for $X_{i-1} \neq \underrightarrow{\#}$ is similar. Furthermore, we can assume that $X_{i+1} \in \{E_R, R, F, \underrightarrow{D}\}$ since if $X_{i+1} \in \{\underrightarrow{W}, \underrightarrow{S}\}$ then s is already in $\mathcal{G}$. We show that the event schema $\mathit{FIRST}((\mathtt{flip}_i, \mathtt{left}), (\mathtt{flip}_{i+1}, \mathtt{right}))$, which by Lemma 6.2.2 has probability at least 1/2, leads eventually to a state of $\mathcal{G} \cup \mathcal{P}$. Let $\mathcal{A}$ be an adversary of *Unit-Time*, and let α be an admissible timed execution of $\Omega_{prexec(M,\{s\},\mathcal{A})}$ where if process i flips before process $i+1$ then process i flips left, and if process $i+1$ flips before process i then process $i+1$ flips right.

Then, within time 1, i performs one transition and reaches W. Let $j \in \{i, i+1\}$ be the first of i and $i+1$ that reaches W and let s_1 be the state reached after the first time process j reaches W. If some process reached P in the meantime, then we are done. Otherwise there are two cases to consider. If $j = i$, then, $\texttt{flip}_i$ gives $\texttt{left}$ and $X_i(s_1) = \underset{\leftarrow}{W}$ whereas X_{i+1} is (still) in $\{E_R, R, F, \underset{\rightarrow}{D}\}$. Therefore, $s_1 \in \mathcal{G}$. If $j = i+1$, then $\texttt{flip}_{i+1}$ gives $\texttt{right}$ and $X_{i+1}(s_1) = \underset{\rightarrow}{W}$ whereas $X_i(s_1)$ is (still) F. Therefore, $s_1 \in \mathcal{G}$. ■

Lemma 10.6.13 *Start with a state s of $\mathcal{F}$. If there exists a process i for which $X_i(s) = F$ and $(X_{i-1}(s), X_{i+1}(s)) = (\underset{\rightarrow}{\#}, \underset{\leftarrow}{\#})$. Then, with probability at least $1/2$, a state of $\mathcal{G} \cup \mathcal{P}$ is reached within time 2.*

Proof. The hypothesis can be summarized into the form $(X_{i-1}(s), X_i(s), X_{i+1}(s)) = (\underset{\rightarrow}{\#}, F, \underset{\leftarrow}{\#})$. Since $i-1$ and $i+1$ point in different directions, by moving to the right of $i+1$ there is a process k pointing to the left such that process $k+1$ either points to the right or is in $\{E_R, R, F, P\}$, i.e., $X_k(s) \in \{\underset{\leftarrow}{W}, \underset{\leftarrow}{S}, \underset{\leftarrow}{D}\}$ and $X_{k+1}(s) \in \{E_R, R, F, \underset{\rightarrow}{W}, \underset{\rightarrow}{S}, \underset{\rightarrow}{D}, P\}$.

If $X_k(s) \in \{\underset{\leftarrow}{W}, \underset{\leftarrow}{S}\}$ and $X_{k+1}(s) \neq P$ then $s \in \mathcal{G}$ and we are done; if $X_{k+1}(s) = P$ then $s \in \mathcal{P}$ and we are done. Thus, we can restrict our attention to the case where $X_k(s) = \underset{\leftarrow}{D}$.

We show that $\mathit{FIRST}((\texttt{flip}_k, \texttt{left}), (\texttt{flip}_{k+1}, \texttt{right}))$, which by Lemma 6.2.2 has probability at least $1/2$, leads to $\mathcal{G} \cup \mathcal{P}$ within time 2. Let $\mathcal{A}$ be an adversary of *Unit-Time*, and let α be an admissible timed execution of $\Omega_{prexec(M,\{s\},\mathcal{A})}$ where if process k flips before process $k+1$ then process k flips left, and if process $k+1$ flips before process k then process $k+1$ flips right.

Within time 2 process k performs at least two transitions and hence goes to configuration W. Let $j \in \{k, k+1\}$ be the first of k and $k+1$ that reaches W and let s_1 be the state reached after the first time process j reaches W. If some process reached P in the meantime, then we are done. Otherwise, we distinguish two cases. If $j = k$, then, $\texttt{flip}_k$ gives $\texttt{left}$ and $X_k(s_1) = \underset{\leftarrow}{W}$ whereas X_{k+1} is (still) in $\{E_R, R, F, \underset{\rightarrow}{\#}\}$. Thus, $s_1 \in \mathcal{G}$. If $j = k+1$, then $\texttt{flip}_{k+1}$ gives $\texttt{right}$ and $X_{k+1}(s_1) = \underset{\rightarrow}{W}$ whereas $X_k(s_1)$ is (still) in $\{\underset{\leftarrow}{D}, F\}$. Thus, $s_1 \in \mathcal{G}$. ■

Proposition 10.6.14 *Start with a state s of $\mathcal{F}$. Then, with probability at least $1/2$, a state of $\mathcal{G} \cup \mathcal{P}$ is reached within time 2. Equivalently:*

$$\mathcal{F} \xrightarrow[1/2]{2} \mathcal{G} \cup \mathcal{P}.$$

Proof. The hypothesis of Lemmas 10.6.12 and 10.6.13 form a partition of $\mathcal{F}$. ■

Finally, we prove $\mathcal{RT} \xrightarrow[1]{} \mathcal{F} \cup \mathcal{G} \cup \mathcal{P}$.

Proposition 10.6.15 *Starting from a state s of $\mathcal{RT}$, then a state of $\mathcal{F} \cup \mathcal{G} \cup \mathcal{P}$ is reached within time 3 Equivalently:*

$$\mathcal{RT} \xrightarrow[1]{3} \mathcal{F} \cup \mathcal{G} \cup \mathcal{P}.$$

Proof. Let s be a state of $\mathcal{RT}$. If $s \in \mathcal{F} \cup \mathcal{G} \cup \mathcal{P}$, then we are trivially done. Suppose that $s \notin \mathcal{F} \cup \mathcal{G} \cup \mathcal{P}$. Then in s each process is in $\{E_R, R, W, S, D\}$ and there exists at least process in $\{W, S, D\}$. Let $\mathcal{A}$ be an adversary of *Unit-Time*, and let α be an admissible timed execution of $\Omega_{prexec(M,\{s\},\mathcal{A})}$.

We first argue that within time 1 some process reaches a state of $\{S, D, F\}$ in α. This is trivially true if in state s there is some process in $\{S, D\}$. If this is not the case, then all processes are either in E_R or R or W. Eventually, some process in R or W performs a transition. If the first process not in E_R performing a transition started in E_R or R, then it reaches F and we are done; if the first process performing a transition is in W, then it reaches S since in s no resource is held. Once a process i is in $\{S, D, F\}$, then within time 2 process i reaches either state F or P, and we are done. ■

10.7 Abstract Complexity Measures

We have seen how to measure the expected time to satisfy a property. However, the technique can be extended to other kinds of measures of complexity. Specifically, let ϕ be a complexity measure on timed execution fragments that is additive under concatenation, i.e., $\phi(q_1 \frown q_2) = \phi(q_1) + \phi(q_2)$. Then we can compute the expected ϕ rather than the expected time, where the ϕ of a state q of H is defined to be $\phi(q \triangleright q_0^H)$. We generalize the notation for timed progress statements by writing

$$U \xrightarrow[p]{\phi(c)}_{Advs} U' \tag{10.34}$$

with the meaning that $\Pr_{Advs,U}(e_{U',\phi(c)}) \geq p$, where the event schema $e_{U',\phi(c)}$ applied to a timed probabilistic execution fragment H returns the set of timed executions α of Ω_H where a state from U' is reached within complexity c. More specifically, let $Cones_{U',\phi(c)}(H)$ be the set of minimal timed execution fragments q of M such that C_q^H is not empty, $lstate(q) \in U'$, and $\phi(q \triangleright q_0^H) \leq c$. Then, $e_{U',\phi(c)}(H) = \cup_{q \in Cones_{U',\phi(c)}(H)} C_q^H$. Observe that time is just one of the possible complexity measures.

The same definition can be extended to sets of actions as we have done previously, and the concatenation theorem is still valid.

The expected complexity of a finitely satisfiable event schema can be defined easily. Specifically, if e is a finitely satisfiable event-schema and $Cones(H)$ identifies the points of satisfaction of e, then for each probabilistic timed execution fragment H of M we define $E_{H,\phi}[e]$, the expected complexity to satisfy e in H, as follows.

$$E_{H,\phi}[e] = \begin{cases} \sum_{q \in Cones(H)} P_H[C_q](\phi(q \triangleright q_0^H)) & \text{if } P_H[e(H)] = 1 \\ \infty & \text{otherwise.} \end{cases} \tag{10.35}$$

Then, a proposition similar to Proposition 10.5.1 can be proved.

Proposition 10.7.1 *Suppose that*

$$\begin{cases} U \xrightarrow[p]{\phi(c)}_{Advs} U' \\ U \Rightarrow (U \; Unless \; U'), \end{cases} \tag{10.36}$$

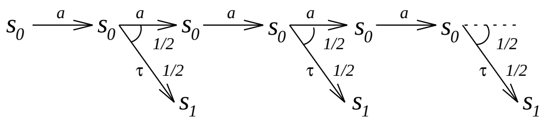


Figure 10-1: An example of the use of ξ.

and suppose that Advs is finite history insensitive and that $s\delta \notin \Omega_{\mathcal{A}(s)}$ *for each* $\mathcal{A} \in Advs$ *and each* $s \in U$. *Then,*

$$E_{U,Advs,\phi}[e] \leq c + pE_{U',Advs,\phi}[e] + (1-p)(\xi + E_{U,Advs,\phi}[e]), \tag{10.37}$$

where

$$\xi = sup_{q \in t\text{-}frag^*(M) | lstate(q) \in U} \left(sup_{q' > q} \left(inf_{q'' | q < q'' \leq q'} (\phi(q'' \triangleright q)) \right) \right). \tag{10.38}$$

Proof. This proof has the same structure as the proof of Proposition 10.5.1. Here we describe in detail only the main differences. In particular, we show part of the derivation from Equation (10.16) to Equation (10.21), where the constant ξ is used. Observe that if we use ϕ to express time complexity, then $\xi = 0$.

From (10.35) the expected complexity for success for e is

$$E_{H,\phi}[e] = \sum_{q \in Cones(H)} P_H[C_q]\phi(q \triangleright q_0^H). \tag{10.39}$$

For each $d > 0$, let $Cones_d$ be a function that expresses the event of reaching complexity d as a union of disjoint cones. From the definition of a probabilistic timed execution, we know that $Cones_d$ exists and, from (10.38), we know that for each probabilistic timed execution fragment H and each $q \in Cones_d(H)$, $d \leq \phi(q \triangleright q_0^H) \leq d + \xi$. Let ϵ be any positive number. Following the same derivation as in the proof of Proposition 10.5.1, we obtain

$$E_{H,\phi}[e] \leq (c+\epsilon)\left(\sum_{q \in \Theta_1} P_H[C_q]E_{U',Advs,\phi}[e]\right) + \left(\sum_{q \in \Theta_2} P_H[C_q](\xi + E_{U,Advs,\phi}[e])\right). \tag{10.40}$$

■

One of the novel aspects of Proposition 10.7.1 is the constant ξ. Roughly speaking, ξ gives us a lower bound to the minimum complexity increase that we can obtain by moving along a timed execution fragment.

Example 10.7.1 (Why ξ is necessary) For example, if the abstract complexity that we use is the number of discrete actions that appear in a timed execution fragment, then $\xi = 1$. In fact, whenever we perform a discrete action, the complexity increases by 1. Figure 10-1 shows an example where $\xi = 1$ and where Equation (10.37) is invalidated if we do not include ξ. Denote the probabilistic timed execution fragment of Figure 10-1 by H. Let U be $\{s_0\}$, U' be $\{s_1\}$, and let e express the property of reaching U'. Let $Advs$ contain only one adversary that generates H when applied to s_0. Let ϕ count the number of external actions in a timed execution fragment (no time-passage actions in H). Then, it is immediate to verify that the statement $U \xrightarrow[1/2]{\phi(1)} U'$ is valid in H and that also $U \Rightarrow (U \; Unless \, U')$ is valid. By applying Equation (10.37) with $\xi = 1$, we obtain

$$E_{U,Advs,\phi}[e] \leq t + 1/2(1 + E_{U,Advs,\phi}[e]), \tag{10.41}$$

which leads to $E_{U,Advs,\phi}[e] \leq 3$. If we did not use ξ in Equation (10.37) we would have obtained $E_{U,Advs,\phi}[e] \leq 2$. We now show that $E_{H,\phi}[e] = 3$. In fact,

$$E_{H,\phi}[e] = \frac{1}{2} + 3\frac{1}{4} + 5\frac{1}{8} + 7\frac{1}{16} + \cdots \tag{10.42}$$

By rearranging the terms, we obtain

$$E_{H,\phi}[e] = \sum_{i \geq 0} \frac{1}{2^i} \left(\frac{1}{2} + \frac{2}{4} + \frac{2}{8} + \frac{2}{16} + \cdots \right). \tag{10.43}$$

Recall that $\sum_{i \geq 0} 1/2^i = 2$. Thus, by rearranging the terms again,

$$E_{H,\phi}[e] = 2 + 1/2 \left(\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \cdots \right) = 3. \tag{10.44}$$

Roughly speaking, the transition relation of H is structured in such a way that whenever the experiment of reaching U' from U fails, the system looses one additional complexity unit during the random draw. In the proof of Proposition 10.7.1 this phenomenon is detected when we define the partition Θ_1 and Θ_2. To make sure that Θ_1 and Θ_2 partition an event with probability 1 and that Θ_1 captures all the places where U' is reached within time t, Θ_2 must be based on states reached after time t. In the probabilistic execution H of this example the states of Θ_2 have complexity $t + 1$. ■

10.8 Example: Randomized Agreement with Time

Using abstract complexity measures it is possible to show that the randomized agreement algorithm of Ben-Or guarantees agreement within an expected exponential time. This is not an exceptional complexity result, but it corresponds to the time complexity of the algorithm.

In more detail, we add time to the probabilistic automaton that describes Ben-Or's protocol in the same way as we have done for the Dining Philosophers algorithm of Lehmann and Rabin. In this case each adversary is required to schedule every process that enables some transition within time 1 from every point. Then we show an upper bound linear in st on the time it takes to all processes to complete a specific stage st. Finally, we derive an upper bound on the expected number of stages it takes for all processes to decide. This is achieved by defining an abstract complexity on the timed executions of M that checks the highest stage reached at every point. A direct extension of the untimed proof without abstract complexities would not be possible. In fact, given a reachable state s, the validity of the progress statement of Chapter 6 relies on completing the highest stage reached in s, and we cannot establish any useful upper bound on the time to complete such stage: there is no useful bound on the difference between the highest and the lowest stages reached in s, and the adversary may stop the processes with the highest values of st. We start by proving the upper bound on the time it takes to each process to complete some stage st.

Lemma 10.8.1 *There is a constant d such that, for each stage st, each process completes stage st within time $d \cdot st$.*

Proof. Let d_1 be the maximum time it takes to each process from the moment it reaches a new stage st to the moment it broadcasts its value and its value is delivered; let d_2 be the maximum time it takes to each process to broadcast and deliver its second message after receiving enough messages from the first round; let d_3 be the maximum time it takes to each process to move to a new stage once it has received enough messages from the second round. Then $d = d_1 + d_2 + d_3$. Since we have not defined formally M, we cannot say explicitly what is the value of d.

We show the result by induction on st where for the base case we assume that $st = 0$ and that stage 0 is completed by time 0. By induction, by time $d \cdot st$ each non-faulty process has completed round st. Then, by time $d_1 + d \cdot st$ each non-faulty process has broadcasted and delivered its first round message, and thus every non-faulty process has received enough messages for the first round of stage $st + 1$. Within additional time d_2 each non-faulty process delivers its second message, and within additional time d_3 each non-faulty process reaches stage $st + 2$, i.e., within time $d(st + 1)$ each non-faulty process completes stage $st + 1$. ■

For each finite timed execution fragment α of M define $\phi(\alpha)$, the stage complexity of α, to be $\textit{max-stage}(\textit{lstate}(\alpha)) - \textit{max-stage}(\textit{fstate}(\alpha))$, where for each state s, $\textit{max-stage}(s)$ is the maximum stage that is reached in s by some process. Observe that this complexity measure is an upper bound to the stage at which some process decides since if at state s the first process has just decided, then $\textit{max-stage}(s)$ is not smaller than the stage of the process that has decided. Thus, an upper bound on the expected ϕ for the decision of the first process is an upper bound on the expected stage at which the first process decides. We show the following two statements.

$$\mathcal{B} \xrightarrow[1]{\phi(1)}_{\textit{f-fair}} \mathcal{F} \cup \mathcal{O}. \tag{10.45}$$

$$\mathcal{F} \xrightarrow[1/2^n]{\phi(2)} \mathcal{O}. \tag{10.46}$$

Then, by combining (10.45) and (10.46) with Theorem 5.5.2, we obtain

$$\mathcal{B} \xrightarrow[1/2^n]{\phi(3)} \mathcal{O}. \tag{10.47}$$

From Proposition 10.7.1, we obtain

$$E_{\mathcal{B},\textit{Unit-Time},\phi}[e_{\mathcal{O}}] \leq 3 + (1 - 1/2^n)(1 + E_{\mathcal{B},\textit{Unit-Time},\phi}[e_{\mathcal{O}}]), \tag{10.48}$$

where 1 is the value of ξ given by (10.38). By solving Equation (10.48) we obtain

$$E_{\mathcal{B},\textit{Unit-Time},\phi}[e_{\mathcal{O}}] \leq 2^{n+2} - 1. \tag{10.49}$$

Since if a process decides at stage st then each other non-faulty process decides within stage $st + 1$, then we can derive that the expected stage by which every process decides is at most 2^{n+2}, and thus, from Lemma 10.8.1, each process decides within expected time $d \cdot 2^{n+1}$.

The proofs for (10.45) and (10.46) have the same structure as the corresponding proofs for the untimed case. Recall that the proof of (10.45) consider the maximum stage st of a reachable state s and states that eventually stage $st + 1$ is reached, at which time a state of $\mathcal{F}$ is reached. The proof of (10.46) states that a specific coin lemma leads a process to decide by stage $\textit{max-stage}(s) + 1$. Then, since if a process decides a stage st each process decides by stage $st + 1$, the complexity of the state where the first process decides is at most $\textit{max-stage}(s) + 2$.

10.9 Discussion

To our knowledge this is the first time that statements similar to our timed progress statements have been used for the analysis of the performance of a randomized distributed algorithm. In particular, we have been able to prove similar results only because we have studied techniques to prove properties that hold with some probability different than 1. This should be a sufficiently strong reason to pursue additional research on methodologies (automatic or not) for the analysis of properties that hold with probabilities different than 1. The work of Hansson [Han94] and the algorithm that Courcoubetis and Yannakakis present in [CY90] are in this direction.

Chapter 11

Hierarchical Verification: Timed Trace Distributions

11.1 Introduction

In this chapter we extend the trace distribution preorder of Chapter 7 to the timed framework. The main difference is that we use *timed traces* rather than traces. A timed trace contains the sequence of discrete actions that occur within a timed execution plus the time of occurrence of each action and the time at which the observation ends. That is, in a timed execution we observe at what time each external action occurs and, if finitely many actions occur, how much time elapses after the occurrence of the last action.

We define a preorder relation based on *timed trace distribution* inclusion, and we characterize the coarsest precongruence that is contained in the timed trace distribution preorder by using a *timed principal context*, which is just the principal context of Chapter 7 augmented with arbitrary time-passage self-loop transitions from its unique state. Most of the proofs follow directly from the results already proved in Chapter 7, since in several cases it is sufficient to study ordinary trace distributions in order to derive properties of timed trace distributions.

11.2 Timed Traces

We start by defining the main object of observation, i.e., timed traces. The definition of a timed trace that we give in this section is taken directly from [LV95].

Timed Sequence Pairs

Let K be any set that does not intersect $\Re^{+}$. Then a *timed sequence* over K is defined to be a (finite or infinite) sequence γ over $K \times \Re^{\geq 0}$ in which the time components are nondecreasing, i.e., if (k,t) and (k',t') are consecutive elements in γ then $t \leq t'$. We say that γ is *Zeno* if it is infinite and the limit of the time components is finite.

A *timed sequence pair* over K is a pair $\beta = (\gamma, t)$, where γ is a timed sequence over K and $t \in \Re^{\geq 0} \cup \{\infty\}$, such that t is greater than or equal to all time components in γ. We write $seq(\beta)$, and $ltime(\beta)$ for the two respective components of β. We denote by $tsp(K)$ the set of timed sequence pairs over K. We say that a timed sequence pair β is *finite* if both $seq(\beta)$ and $ltime(\beta)$ are finite, and *admissible* if $seq(\beta)$ is not Zeno and $ltime(\beta) = \infty$.

Let β and β' be timed sequence pairs over K with β finite. Then define $\beta;\beta'$ to be the timed sequence pair $(seq(\beta)\gamma, ltime(\beta) + ltime(\beta'))$, where γ is the modification of $seq(\beta')$ obtained by adding $ltime(\beta)$ to all the time components. If β and β' are timed sequence pairs over a set K, then β is a *prefix* of β', denoted by $\beta \leq \beta'$, if either $\beta = \beta'$, or β is finite and there exists a timed sequence pair β'' such that $\beta' = \beta;\beta''$.

Lemma 11.2.1 *$\leq$ is a partial ordering on the set of timed sequence pairs over K.* ■

Now we describe how to translate from a sequence over $K \cup \Re^+$, and ordinary trace, to a timed sequence pair over K. First, if β is any sequence over $K \cup \Re^+$, then we define the *time of occurrence* of any K-element in β to be the sum of all the reals that precede that element in β. We also define $ltime(\beta)$ to be the sum of all the reals in β. Finally, we define *t-trace*(β) to be the timed sequence pair $(\gamma, ltime(\beta))$, where γ is the subsequence of β consisting of all the elements of K, each paired with its time of occurrence.

If β is a sequence over $K \cup \Re^+$ then we say that β is *admissible* if the sum of the positive reals in β is infinite.

Lemma 11.2.2 *If β is a finite or admissible timed sequence pair then t-trace$(trace(\beta)) = \beta$.* ■

Lemma 11.2.3 *If β is a sequence over $K \cup \Re^+$ then β is admissible if and only if t-trace(β) is admissible.* ■

Timed Traces of Timed Probabilistic Automata

Suppose that $\alpha = \omega_0 a_1 \omega_1 a_2 \omega_2 \cdots$ is a timed execution fragment of a timed probabilistic automaton M. For each a_i, define the *time of occurrence* t_i to be $\sum_{j<i} ltime(\omega_j)$, i.e., the sum of the lengths of all the trajectory intervals preceding a_i in α. Let γ be the sequence consisting of the actions in α paired with their times of occurrence:

$$\gamma = (a_1, t_1)(a_2, t_2)\cdots.$$

Then *t-trace*(α), the *timed trace* of α, is defined to be the pair

$$(\gamma \upharpoonright (vis(M) \times \Re^+), ltime(\alpha)).$$

Thus, *t-trace*(α) records the occurrences of visible actions together with their times of occurrence, and together with the time spanned by α. Note that neither internal actions nor time-passage actions appear explicitly in the timed trace of α.

Proposition 11.2.4 *If α is a timed execution fragment of M then t-trace(α) is a timed sequence pair over $vis(M)$.* ■

Proposition 11.2.5 *Let α be a timed execution fragment of M, and let trace(α) denote the ordered sequence of external actions that appear in α. Then, t-trace(α) = t-trace$(trace(\alpha))$.* ■

Proposition 11.2.6 *If $\alpha = \alpha_1 \frown \alpha_2$ is a timed execution fragment of M, then t-trace(α) = t-trace(α_1); t-trace(α_2).* ■

We write *t-traces*(M) for the set of all timed traces of M, *t-traces*$^*(M)$ for the set of *finite* timed traces of M, and *t-traces*$^\infty(M)$ for the set of *admissible* timed traces of M,

The timed traces of a probabilistic timed automaton M can be characterized also in terms of its time-enriched executions or in terms of its ordinary executions. Specifically, if α is a time-enriched execution of M, then let *t-trace*(α) denote *t-trace*(*t-exec*(α)), and if α is an execution of M, then let *t-trace*(α) denote *t-trace*(*trace*(α)). The following proposition holds.

Proposition 11.2.7 *Let M be a probabilistic timed automaton.*

1. *If α is a time-enriched execution of M, then there is a timed execution α' of M such that t-trace(α) = t-trace(α').*

2. *If α is a timed execution of M, then there is a time-enriched execution α' of M such that t-trace(α) = t-trace(α').*

3. *If α is a timed execution of M, then there is an execution α' of M such that t-trace(α) = t-trace(α').*

4. *If α is an execution of M, then there is a timed execution α' of M such that t-trace(α) = t-trace(α').*

Proof.

1. Let α' be *t-exec*(α). Then, *t-trace*(α) = *t-trace*(α') by definition.

2. Let α be $\omega_0 a_1 \omega_1 a_2 \cdots$. If α is a finite timed execution or an infinite sequence, then let $\alpha' = \mathit{fstate}(\omega_0) \frown \alpha_1 \frown \alpha_2 \frown \cdots$, where for each i,

$$\alpha_i = \begin{cases} \omega_{i-1} a_i \mathit{fstate}(\omega_i) & \text{if } \omega_{i-1} \text{ has domain } [0,0], \\ \mathit{fstate}(\omega_{i-1}) \mathit{ltime}(\omega_{i-1}) \omega_{i-1} a_i \mathit{fstate}(\omega_i) & \text{otherwise;} \end{cases}$$

   if $\alpha = \omega_0 a_1 \omega_1 a_2 \cdots a_n \omega_n$ and the domain of ω_n is right-open, then let $\alpha' = \mathit{fstate}(\omega_0) \frown \alpha_1 \frown \cdots \frown \alpha_n \frown \alpha'_{n+1}$, where the α_i's are defined above and $\alpha'_{n+1} = \omega'_0 d_1 \omega'_1 d_2 \omega'_2 \cdots$ is an infinite sequence such that $\omega'_0 \omega'_1 \omega'_2 \cdots = \omega_n$. It is immediate to verify that α and α' have the same timed trace since α = *t-exec*(α').

3. Let α be $\omega_0 a_1 \omega_1 a_2 \cdots$. If α is a finite timed execution or an infinite sequence, then let $\alpha' = \mathit{fstate}(\omega_0) \frown \alpha_1 \frown \alpha_2 \frown \cdots$, where for each i,

$$\alpha_i = \begin{cases} \mathit{lstate}(\omega_{i-1}) a_i \mathit{fstate}(\omega_i) & \text{if } \omega_{i-1} \text{ has domain } [0,0], \\ \mathit{fstate}(\omega_{i-1}) \mathit{ltime}(\omega_{i-1}) \mathit{lstate}(\omega_{i-1}) a_i \mathit{fstate}(\omega_i) & \text{otherwise;} \end{cases}$$

   if $\alpha = \omega_0 a_1 \omega_1 a_2 \cdots a_n \omega_n$ and the domain of ω_n is right-open, then let $\alpha'' = \mathit{fstate}(\omega_0) \frown \alpha_1 \frown \cdots \frown \alpha_n \frown \alpha'_{n+1}$, where the α_i's are defined above and $\alpha'_{n+1} = \mathit{fstate}(\omega_n) d_1 \omega_n(d_1) d_2 \omega_n(d_1 + d_2) \cdots$ is an infinite sequence such that $\sum_i d_i = \mathit{ltime}(\omega_n)$. It is immediate to verify that α and α' have the same timed trace.

4. Given $\alpha = s_0 a_1 s_1 a_2 \cdots$, build a time-enriched execution α'' by replacing each state s_i with a trajectory for (s_{i-1}, a_i, s_i) whenever a_i is a time-passage action. Then, $\textit{t-trace}(\alpha) = \textit{t-trace}(\alpha'')$. Item 2 is enough to conclude. ■

The bottom line of the proposition above is that for the study of the timed traces of a probabilistic timed automaton it is not necessary to observe the trajectories spanned by a computation. The points of occurrence of discrete actions are sufficient.

11.3 Timed Trace Distributions

In this section we define the timed trace distributions of a probabilistic timed automaton and we extend the action restriction operation. The main result is that it is possible to study the timed trace distributions of a probabilistic timed automaton M by considering either its probabilistic executions, or its probabilistic time-enriched executions, or its probabilistic timed executions.

11.3.1 Three ways to Define Timed Trace Distributions

We now define the timed trace distribution of a probabilistic execution, of a probabilistic time-enriched execution, and of a probabilistic timed execution of a probabilistic timed automaton. The definitions are given in the same style as for the untimed case. Furthermore, we show that the three definitions lead to the same collection of timed trace distributions. This enforces the remark that for the study of the timed trace distributions of a probabilistic timed automaton it is not necessary to observe the trajectories spanned by a computation.

Timed Trace Distribution of a Probabilistic Execution

Let H be a probabilistic execution of a probabilistic timed automaton M, and let f be a function from Ω_H to $\Omega = \textit{tsp}(\textit{vis}(M))$ that assigns to each extended execution its timed trace. The *timed trace distribution* of H, denoted by $\textit{t-tdistr}(H)$, is the probability space $\textit{completion}((\Omega, \mathcal{F}, P))$ where $\mathcal{F}$ is the σ-field generated by the cones C_β, where β is a finite timed sequence pair of $\textit{tsp}(\textit{vis}(M))$, and $P = f(P_H)$. Note that from Proposition 3.1.4 f is a measurable function from $(\Omega_H, \mathcal{F}_H)$ to $(\Omega, \mathcal{F})$.

Timed Trace Distribution of a Probabilistic Time-Enriched Execution

Let H be a probabilistic time-enriched execution of a probabilistic timed automaton M, and let f be a function from Ω_H to $\Omega = \textit{tsp}(\textit{vis}(M))$ that assigns to each time-enriched extended execution its timed trace. The *timed trace distribution* of H, denoted by $\textit{t-tdistr}(H)$, is the probability space $(\Omega, \mathcal{F}, P)$ where $\mathcal{F}$ is the σ-field generated by the cones C_β, where β is a finite timed timed sequence pair of $\textit{tsp}(\textit{vis}(M))$, and $P = f(P_H)$. Note that from Proposition 3.1.4 f is a measurable function from $(\Omega_H, \mathcal{F}_H)$ to $(\Omega, \mathcal{F})$.

Timed Trace Distribution of a Probabilistic Timed Execution

Let H be a probabilistic timed execution of a probabilistic timed automaton M, and let f be a function from Ω_H to $\Omega = \textit{tsp}(\textit{vis}(M))$ that assigns to each timed extended execution its timed trace. The *timed trace distribution* of H, denoted by *t-tdistr*(H), is the probability space $(\Omega, \mathcal{F}, P)$ where $\mathcal{F}$ is the σ-field generated by the cones C_β, where β is a finite timed timed sequence pair of *tsp*(*vis*(M)), and $P = f(P_H)$. Note that from Proposition 3.1.4 f is a measurable function from $(\Omega_H, \mathcal{F}_H)$ to $(\Omega, \mathcal{F})$.

Equivalence of the Definitions

We now show that the three definitions of a timed trace distribution lead to the same collection of timed trace distributions when applied to a probabilistic timed automaton (cf. Propositions 11.3.2 and 11.3.4). Thus, we can freely denote a generic timed trace distribution by $\mathcal{D}$ and denote the timed trace distributions of a probabilistic tomed automaton M by *t-tdistrs*(M).

Lemma 11.3.1 *Let H be a probabilistic time-enriched execution of a probabilistic timed automaton M. Then, t-tdistr(H) = t-tdistr(sample(H)).*

Proof. Let $\mathcal{D}$ be *t-tdistr*(H) and let $\mathcal{D}'$ be *t-tdistr*(*sample*(H)) Consider a finite timed trace β. From the definition of *t-tdistr*(),

$$P_{\mathcal{D}'}[C_\beta] = P_{sample(H)}[\{\alpha \in \Omega_{sample(H)} \mid \beta \leq \textit{t-trace}(\alpha)\}]. \tag{11.1}$$

Since C_β is a finitely satisfiable event, there is a set of Θ of states of *sample*(H) such that for each element q of Θ, $\beta \leq \textit{t-trace}(q)$, and such that

$$\{\alpha \in \Omega_{sample(H)} \mid \beta \leq \textit{t-trace}(\alpha)\} = \cup_{q \in \Theta} C_q^{sample(H)}. \tag{11.2}$$

Thus,

$$P_{\mathcal{D}'}[C_\beta] = \sum_{q \in \Theta} P_{sample(H)}[C_q^{sample(H)}]. \tag{11.3}$$

From Equation (9.55), Equation (11.3) becomes

$$P_{\mathcal{D}'}[C_\beta] = \sum_{q \in sample^{-1}(\Theta)} P_H[C_q^H]. \tag{11.4}$$

Observe that *sample*$^{-1}(\Theta)$ is a characterization of C_β for $\mathcal{D}$, and thus,

$$P_{\mathcal{D}'}[C_\beta] = P_{\mathcal{D}}[C_\beta]. \tag{11.5}$$

This completes the proof. ■

Proposition 11.3.2 *Let M be a probabilistic timed automaton. Then, for each probabilistic time-enriched execution H of M there exists a probabilistic execution H' of M such that t-tdistr(H) = t-tdistr(H'), and for each probabilistic execution H of M there exists a probabilistic time-enriched execution H' of M such that t-tdistr(H) = t-tdistr(H').*

Proof. Follows directly from Propositions 9.3.6 and 9.3.7, and from Lemma 11.3.1. ■

Lemma 11.3.3 *Let H be a probabilistic time-enriched execution of a probabilistic timed automaton M. Then, t-tdistr(H) = t-tdistr(t-sample(H)).*

Proof. Let $\mathcal{D}$ be *t-tdistr*(H), and let $\mathcal{D}'$ be *t-tdistr*(*t-sample*(H)). Consider a finite timed sequence pair $\mathcal{D}$ of *tsp*(*vis*(M)). From the definition of *t-tdistr*,

$$P_{\mathcal{D}}[C_\beta] = P_H[\{\alpha \in \Omega_H \mid \beta \leq \textit{t-trace}(\alpha)\}]. \tag{11.6}$$

From the definition of *t-exec*$(\mathcal{P}_H)$,

$$P_{\mathcal{D}}[C_\beta] = P_{\textit{t-exec}(\mathcal{P}_H)}[\{\alpha \in \Omega_{\textit{t-exec}(H)} \mid \beta \leq \textit{t-trace}(\alpha)\}]. \tag{11.7}$$

With a similar analysis,

$$P_{\mathcal{D}'}[C_\beta] = P_{\textit{t-sample}(H)}[\{\alpha \in \Omega_{\textit{t-sample}(H)} \mid \beta \leq \textit{t-trace}(\alpha)\}]. \tag{11.8}$$

Since from Proposition 9.3.11 *t-exec*$(\mathcal{P}_H) = \mathcal{P}_{\textit{t-sample}(H)}$, and since the events of (11.7) and (11.8) are unions of countably many disjoint cones, we conclude that $P_{\mathcal{D}}[C_\beta] = P_{\mathcal{D}'}[C_\beta]$. ■

Proposition 11.3.4 *Let M be a probabilistic timed automaton. Then, for each probabilistic time-enriched execution H of M there exists a probabilistic timed execution H' of M such that t-tdistr(H) = t-tdistr(H'), and for each probabilistic timed execution H of M there exists a probabilistic time-enriched execution H' of M such that t-tdistr(H) = t-tdistr(H').*

Proof. Follows directly from Propositions 9.3.8 and 9.3.9, and from Lemma 11.3.3. ■

Proposition 11.3.5 *Let H_1 and H_2 be two equivalent probabilistic time-enriched executions of a probabilistic timed automaton M. Then, t-tdistr(H_1) = t-tdistr(H_2).*

Proof. From Proposition 9.3.10, *t-sample*(H_1) = *t-sample*(H_2), and from Lemma 11.3.3, *tdistr*(H_1) = *tdistr*(*t-sample*(H_1)) and *tdistr*(H_2) = *tdistr*(*t-sample*(H_2)). Thus, combining the observations above, *t-tdistr*(H_1) = *t-tdistr*(H_2). ■

11.3.2 Timed Trace Distribution of a Trace Distribution

Given a trace distribution of a probabilistic timed automaton, it is possible to define its timed trace distribution as we have done for ordinary traces. Thus, let $\mathcal{D}$ be a trace distribution of a probabilistic automaton, and let f be a function from $\Omega_{\mathcal{D}}$ to $\Omega = \{\textit{t-trace}(\beta) \mid \beta \in \Omega_{\mathcal{D}}\}$ that assigns to each trace its timed trace. The *timed trace distribution* of $\mathcal{D}$, denoted by *t-tdistr*$(\mathcal{D})$, is the probability space *completion*$((\Omega, \mathcal{F}, P))$ where $\mathcal{F}$ is the σ-field generated by the cones C_β, where β is a finite timed trace, and $P = f(P_{\mathcal{D}})$. Note that from Proposition 3.1.4 f is a measurable function from $(\Omega_{\mathcal{D}}, \mathcal{F}_{\mathcal{D}})$ to $(\Omega, \mathcal{F})$.

Proposition 11.3.6 *Let H be a probabilistic execution of a timed probabilistic automaton M. Then, t-tdistr(H) = t-tdistr(tdistr(H)).*

Proof. Let $\mathcal{D}$ be *t-tdistr*(H), and let $\mathcal{D}'$ be *t-tdistr*(*tdistr*(H)). We show first that $\mathcal{D}$ and $\mathcal{D}'$ have the same sample space. Then, we show that they assign the same probability to each cone.

To show that $\mathcal{D}$ and $\mathcal{D}'$ have the same sample space, it is enough to show that for each timed sequence pair β of *tsp*(*vis*(M)) thehre is a trace β' of *ext*$(M)^* \cup$ *ext*$(M)^\omega$ such that *t-trace*$(\beta') = \beta$. Let $(\beta = (a_1, t_1)(a_2, t_2), (a_3, t_3) \cdots, t)$. If *seq*$(\beta)$ is an infinite sequence, then let $\beta' = \beta_1\beta_2\beta_3\cdots$, where for each i, if $t_{i+1} = t_i$, then $\beta_i = a_i$, and if $t_{i+1} > t_i$, then $\beta_i =$

$a_i(t_{i+1} - t_i)$. If $seq(\beta)$ is a finite sequence, i.e., $seq(\beta) = (a_1, t_1)(a_2, t_2), (a_3, t_3) \cdots, (a_n, t_n)$ then $\beta' = \beta_1\beta_2\beta_3 \cdots \beta_{n-1}\beta'_n$ where the β_i's are defined above, and β'_n is a_n if $t_n = t$, $a_n(t - t_n)$ if $0 < t - t_n < \infty$, and a_n followed by the infinite sequence of 1's if $t = \infty$. It is easy to verify that in every case $t\text{-}trace(\beta') = \beta$.

To show that $\mathcal{D}$ and $\mathcal{D}'$ assign the same probability to each cone, let β be a finite timed trace. From the definition of *t-tdistr* and *tdistr*,

$$P_{\mathcal{D}'}[C_\beta] = P_H[\{\alpha \in \Omega_H \mid \beta \leq t\text{-}trace(trace(\alpha))\}]. \tag{11.9}$$

From Proposition 11.2.5, (11.9) becomes

$$P_{\mathcal{D}'}[C_\beta] = P_H[\{\alpha \in \Omega_H \mid \beta \leq t\text{-}trace(\alpha)\}], \tag{11.10}$$

which is the definition of $P_{\mathcal{D}}[C_\beta]$. ■

11.3.3 Action Restriction

Finally, we extend the action restriction operator to timed trace distributions. Let M be a probabilistic timed automaton, and let V be a set of visible actions of M. For each timed trace $\beta = (\gamma, t)$ of M, let $\beta \upharpoonright V$ be the pair (γ', t) where γ' is obtained from γ by removing all the pairs whose action is in V. Let $\mathcal{D}$ be a timed trace distribution of M. Define $\mathcal{D} \upharpoonright V$ to be the timed trace distribution $(\Omega, \mathcal{F}, P)$ where $\Omega = \Omega_{\mathcal{D}} \upharpoonright V$, $\mathcal{F}$ is the σ-field generated by the cones C_β, where β is a finite timed trace, and $P = P_{\mathcal{D}} \upharpoonright V$. Note that from Proposition 3.1.4 $\upharpoonright V$ is a measurable function from $(\Omega_{\mathcal{D}}, \mathcal{F}_{\mathcal{D}})$ to $(\Omega, \mathcal{F})$. Action restriction commutes with the operation of taking a timed trace distribution of a trace distribution.

Proposition 11.3.7 *Let $\mathcal{D}$ be a trace distribution of a probabilistic timed automaton M, and let V be a set of visible actions of M. Then, t-tdistr($\mathcal{D} \upharpoonright V$) = t-tdistr($\mathcal{D}$) $\upharpoonright V$.*

Proof. Let $\mathcal{D}'$ be *t-tdistr*($\mathcal{D} \upharpoonright V$), and let $\mathcal{D}''$ be *t-tdistr*($\mathcal{D}$) $\upharpoonright V$. Let β be a finite timed trace. By applying the definitions of *t-tdistr* and of $\upharpoonright$, we obtain the following two equations.

$$P_{\mathcal{D}'}[C_\beta] = P_{\mathcal{D}}[\{\beta' \in \Omega_{\mathcal{D}} \mid \beta \leq t\text{-}trace(\beta' \upharpoonright V)\}]. \tag{11.11}$$

$$P_{\mathcal{D}''}[C_\beta] = P_{\mathcal{D}}[\{\beta' \in \Omega_{\mathcal{D}} \mid \beta \leq t\text{-}trace(\beta') \upharpoonright V\}]. \tag{11.12}$$

Observe that for each β' of $\Omega_{\mathcal{D}}$, $t\text{-}trace(\beta' \upharpoonright V) = t\text{-}trace(\beta') \upharpoonright V$. Thus, the right expressions of (11.11) and (11.12) denote the same value. That is, $P_{\mathcal{D}'}[C_\beta] = P_{\mathcal{D}''}[C_\beta]$. ■

11.4 Timed Trace Distribution Precongruence

Let M_1, M_2 be two probabilistic timed automata with the same external actions. The *timed trace distribution preorder* is defined as follows.

$$M_1 \sqsubseteq_{Dt} M_2 \text{ iff } t\text{-}tdistrs(M_1) \subseteq t\text{-}tdistrs(M_2).$$

As for the untimed case, the timed trace distribution preorder is not a precongruence. A counterexample can be created directly from the counterexample of Chapter 7 by augmenting the probabilistic automata of Figure 7-4 with arbitrary self-loop time-passage transitions from their deadlock states (the states that do not enable any transition). Thus, we define the *timed trace distribution precongruence*, denoted by $\sqsubseteq_{DCt}$, as the coarsest precongruence that is contained in the timed trace distribution preorder.

11.5 Alternative Characterizations

The timed trace distribution precongruence can be characterized by a timed version of the principal context of Chapter 7. Namely, let the *timed principal context*, denoted by C_P be the principal context of Figure 7-6 augmented with self-loop time-passage transitions for each time-passage action d. Then, the following holds.

Theorem 11.5.1 *$M_1 \sqsubseteq_{DCt} M_2$ iff $M_1 \| C_P \sqsubseteq_{Dt} M_2 \| C_P$.*

Thus, if we define the *principal timed trace distributions* of a probabilistic timed automaton M, denoted by $pt\text{-}tdistrs(M)$, to be the timed trace distributions of $M \| C_P$, then we get the following.

Corollary 11.5.2 *$M_1 \sqsubseteq_{DCt} M_2$ iff $ext(M_1) = ext(M_2)$ and $pt\text{-}tdistrs(M_1) \subseteq pt\text{-}tdistrs(M_2)$.*
■

The rest of this section is dedicated to the proof of Theorem 11.5.1. The structure of the proof follows the same lines as the proof of Theorem 7.5.1, where only one additional transformation step is added: a distinguishing context is transformed into a new *time-deterministic* context where each state enables either discrete actions only or time-passage actions only. A time-deterministic context is a probabilistic automaton such that for each state s and each time-passage action d, if $s \xrightarrow{d} s_1$ and $s \xrightarrow{d} s_2$, then $s_1 = s_2$. All the lemmas except for one are proved by reducing the problem to the untimed framework.

Lemma 11.5.3 *Let C be a distinguishing context for two probabilistic timed automata M_1 and M_2. Then there exists a distinguishing context C' for M_1 and M_2 with no discrete actions in common with M_1 and M_2. C' is called a* separated context.

Proof. The context C' is built from C in the same way as in the proof of Lemma 7.5.3. The constructions *clp* and *exch* work as well (they never exchange transitions involving time-passage), and the proof is carried out at the level of probabilistic executions rather than probabilistic timed executions.

Specifically, let $\mathcal{D}$ be a timed trace distribution of $M_1 \| C$ that is not a timed trace distribution of $M_2 \| C$. Consider a probabilistic execution H_1 of $M_1 \| C$ such that $t\text{-}tdistr(H_1) = \mathcal{D}$, and consider the scheduler that leads to H_1. Apply to $M_1 \| C'$ the same scheduler with the following modification: whenever a transition $((s_1, c), a, \mathcal{P}_1 \otimes \mathcal{P})$ is scheduled in $M_1 \| C$, schedule $((s_1, c), a_1, \mathcal{D}((s_1, c')))$, where c' is $c_{(c,a,\mathcal{P})}$, followed by $((s_1, c'), a, \mathcal{P}_1 \otimes \mathcal{D}(c'))$, and, for each $s_1' \in \Omega_1$, followed by $((s_1', c'), a_2, \mathcal{D}(s_1') \otimes \mathcal{P})$. Denote the resulting probabilistic execution by H_1' and the resulting timed trace distribution by $\mathcal{D}'$. From Lemma 7.5.3, $tdistr(H_1) = tdistr(H_1') \upharpoonright vis(M_1 \| C)$, and thus, from Propositions 11.3.6 and 11.3.7, $\mathcal{D} = \mathcal{D}' \upharpoonright vis(M_1 \| C)$.

Suppose by contradiction that it is possible to obtain $\mathcal{D}'$ from $M_2 \| C'$. Consider the scheduler that leads to $\mathcal{D}'$ in $M_2 \| C'$, and let H_2' be the corresponding probabilistic execution. Then, from Lemma 7.5.3, $clp(exch(H_2'))$ is a probabilistic execution of $M_2 \| C'$, and $tdistr(clp(exch(H_2'))) = tdistr(H_2') \upharpoonright acts(M_1 \| C)$. From Propositions 11.3.6 and 11.3.7, $\mathcal{D} = t\text{-}tdistr(clp(exch(H_2')))$, which is a contradiction. ■

Lemma 11.5.4 *Let C be a distinguishing separated context for two probabilistic timed automata M_1 and M_2. Then there exists a distinguishing cycle-free separated context C' for M_1 and M_2.*

Proof. The context C' can be built by unfolding C. Every scheduler for C can be transformed into a scheduler for C' and vice versa, leading to the same timed trace distributions. ∎

Lemma 11.5.5 *Let C be a distinguishing cycle-free, separated context for two probabilistic timed automata M_1 and M_2. Then there exists a distinguishing time-deterministic, cycle-free separated context C' for M_1 and M_2 that from any state enables either time-passage actions only or discrete actions only.*

Proof. The context C' is built from C as follows:

1. for each time-passage transition $s \xrightarrow{d} s'$ of C and each trajectory ω for $s \xrightarrow{d} s'$, add an action $start_\omega$ and an action end_ω;
2. for each time-passage transition $s \xrightarrow{d} s'$ of C and each trajectory ω for $s \xrightarrow{d} s'$, add a collection of new states $\{s_{\omega,t} \mid 0 \leq t \leq d\}$, a transition $s \xrightarrow{start_\omega} s_{\omega,0}$, a transition $s_{\omega,d} \xrightarrow{end_\omega} s'$, and for each $0 \leq t < t' \leq d$, a transition $s_{\omega,t} \xrightarrow{t'-t} s_{\omega,t'}$;
3. remove all the time-passage transitions leaving from states of C.

Let $\mathcal{D}$ be a timed trace distribution of $M_1\|C$ that is not a timed trace distribution of $M_2\|C$. Consider a probabilistic execution H_1 of $M_1\|C$ such that $\textit{t-tdistr}(H_1) = \mathcal{D}$, and consider the scheduler that leads to H_1. Apply to $M_1\|C'$ the same scheduler with the following modification: whenever a time-passage transition $s \xrightarrow{d} s'$ is scheduled, choose a trajectory ω for $s \xrightarrow{d} s'$ and schedule $start_\omega$, followed by d, and followed by end_ω. Denote the resulting probabilistic execution by H_1' and the resulting timed trace distribution by $\mathcal{D}'$. Then,

$$\mathcal{D}' \upharpoonright acts(M_1\|C) = \mathcal{D}. \tag{11.13}$$

To prove (11.13) we prove first that $tdistr(H_1') \upharpoonright acts(M_1\|C) = tdistr(H_1)$, and then we apply Propositions 11.3.6 and 11.3.7. To prove that $tdistr(H_1') \upharpoonright acts(M_1\|C) = tdistr(H_1)$ we define a construction $tclp$ to be applied to probabilistic executions of $M_i\|C'$ where each occurrence of a start action is followed eventually by the corresponding end action with probability 1.

Let H' be a probabilistic execution of $M_i\|C'$ where each occurrence of a start action is followed eventually by the corresponding end action with probability 1, and denote $tclp(H')$ by H. For each state q of H', let $tclp(q)$ be obtained from q by replacing each state of the form $s_{\omega,t}$ with the state $\omega(t)$, by removing each occurrence of a start action together with its following state, and by removing each end action together with its following state. Then,

$$states(H) \triangleq tclp(states(H')). \tag{11.14}$$

Let $(q, \mathcal{P})$ be a restricted transition of H', and suppose that no start or end action occurs. Let $\Omega' = \{(a, tclp(q')) \mid (a, q') \in \Omega\}$, and for each $(a, q'') \in \Omega'$, let $P'[(a, q'')] = P[a \times tclp^{-1}(q'')]$, where $tclp^{-1}(q)$ is the set of states q' of H' such that $tclp(q') = q$. Then the transition $tclp((q, \mathcal{P}))$ is defined to be

$$tclp((q, \mathcal{P})) \triangleq (tclp(q), \mathcal{P}). \tag{11.15}$$

For the transition relation of H, consider a state q of H, and let $min(tclp^{-1}(q))$ be the set of minimal states of $tclp^{-1}(q)$ under prefix ordering. For each state $\bar{q} \in tclp^{-1}(q)$, let

$$\bar{p}_{\bar{q}}^{tclp^{-1}(q)} \triangleq \frac{P_{H'}[C_{\bar{q}}]}{\sum_{q' \in min(tclp^{-1}(q))} P_{H'}[C_{q'}]}. \tag{11.16}$$

The transition enabled from q in H is

$$\sum_{q' \in tclp^{-1}(q)} \bar{p}_{\bar{q}}^{tclp^{-1}(q)} P_{q'}^{H'}[acts(M_i \| C)] tclp(tr_{q'}^{H'} \restriction acts(M_i \| C)). \tag{11.17}$$

The probabilistic execution H satisfies the following properties.

a. H is a probabilistic execution of $M_i \| C$.

The fact that each state of H is reachable can be shown by a simple inductive argument; the fact that each state of H is a finite execution fragment of $M_i \| C$ follows from a simple analysis of the definition of $tclp$.

From (11.17) it is enough to check that for each state q' of H', the transition $tclp(tr_{q'}^{H'} \restriction acts(M_i \| C))$ is generated a combined transition of $M_i \| C$. Since $tr_{q'}^{H'}$ is a transition of H', $(tr_{q'}^{H'} \restriction acts(M_i \| C))$ can be expressed as $q' \frown tr$, where tr is a combined transition of $M_i \| C'$ and no start or end action occurs in tr. Let tr' be obtained by substituting each state of the form $s_{\omega,t}$ with $\omega(t)$ in tr. Then, tr' is a combined transition of $M \| C$, and, from the definition of $tclp$, $tclp(tr_{q'}^{H'} \restriction acts(M_i \| C)) = tclp(q') \frown tr'$.

b. For each state q of H,

$$P_H[C_q] = \sum_{q' \in min(tclp^{-1}(q))} P_{H'}[C_{q'}]. \tag{11.18}$$

This is shown by induction on the length of q. If q consists of a start state only, then the result is trivial. Otherwise, from the definition of the probability of a cone, Equation (11.17), and a simple algebraic simplification,

$$P_H[C_{qas}] = P_H[C_q] \left(\sum_{q' \in tclp^{-1}(q)} \bar{p}_{q'}^{tclp^{-1}(q)} P_{q'}^{H'}[a \times tclp^{-1}(qas)] \right). \tag{11.19}$$

Observe that for each $q' \in tclp^{-1}(q)$ the set $\Omega_{q'}^{H'} \cap (\{a\} \times tclp^{-1}(qas))$ contains only one element, say $(a, q'as'')$, and thus $P_{H'}[C_{q'}] P_{q'}^{H'}[a \times tclp^{-1}(qas)]$ gives $P_{H'}[C_{q'as''}]$. Moreover, observe that the states of $min(tclp^{-1}(qas))$ are the states of the form described in Equation (11.19) (simple cases analysis). Thus, by applying induction to (11.19), using (11.16), simplifying algebraically, and using the observations above,

$$P_H[C_{qas}] = \sum_{q' \in min(tclp^{-1}(qas))} P_{H'}[C_{q'}]. \tag{11.20}$$

c. $tdistr(H) = tdistr(H') \upharpoonright acts(M_i \| C)$.

Let β be a finite trace of H or H'. Then $\{\alpha \in \Omega_{H'} \mid \beta \leq trace(\alpha) \upharpoonright acts(M_i \| C)\}$ can be expressed as a union of disjoint cones $\cup_{q \in \Theta} C_q$ where

$$\Theta = \{q \in states(H') \mid trace(q) \upharpoonright acts(M_i \| C) = \beta, lact(q) = lact(\beta)\}. \qquad (11.21)$$

The set $tclp(\Theta)$ is the set

$$tclp(\Theta) = \{q \in states(H) \mid trace(q) = \beta, lact(q) = lact(\beta)\}, \qquad (11.22)$$

which is a characterization of $\{\alpha \in \Omega_H \mid \beta \leq trace(\alpha)\}$ as a union of disjoint cones. Observe that $min(tclp^{-1}(tclp(\Theta))) = \Theta$. Moreover, for each $q_1 \neq q_2$ of $tclp(\Theta)$, $tclp^{-1}(q_1) \cap tclp^{-1}(q_2) = \emptyset$. Thus, from (11.18), $P_{H'}[\cup_{q \in \Theta} C_q] = P_H[\cup_{q \in tclp(\Theta)} C_q]$. This is enough to conclude.

To complete the proof of (11.13) it is enough to observe that $H_1 = tclp(H_1')$. Property (11.13) is then expressed by property (c).

Suppose by contradiction that it is possible to obtain $\mathcal{D}'$ from $M_2 \| C'$. Consider the scheduler that leads to $\mathcal{D}'$ in $M_2 \| C'$, and let H_2' be the corresponding probabilistic execution. Observe that, since the timed trace distribution of H_2' is $\mathcal{D}'$, and since by construction in $\mathcal{D}'$ each occurrence of a start action is followed eventually by the corresponding end action with probability 1, in H_2' each occurrence of a start action is followed eventually by the corresponding end action with probability 1. Thus, $tclp$ can be applied, and $t\text{-}tdistr(tclp(H_2')) = \mathcal{D}$, which is a contradiction. ■

Lemma 11.5.6 *Let C be a distinguishing time-deterministic, cycle-free, separated context for two probabilistic timed automata M_1 and M_2 that from any state enables either time-passage actions only or discrete actions only. Then there exists a distinguishing time-deterministic, cycle-free separated context C' for M_1 and M_2 that from any state enables either time-passage actions only or discrete actions only, and such that the transition relation from any state enabling discrete actions is at most countably branching. C' is called a* time-deterministic, countably-branching, cycle-free separated context.

Proof. Let $\mathcal{D}$ a timed trace distribution of $M_1 \| C$ that is not a timed trace distribution of $M_2 \| C$. Consider one of the corresponding probabilistic executions H. Observe that H has at most countably many states that enable discrete actions, and that at each state of H there are at most countably many transitions of C that are scheduled. Thus, in total, only countably many discrete transitions of C are used to generate $\mathcal{D}$. Then C' is C without the useless discrete transitions. ■

Lemma 11.5.7 *Let C be a distinguishing time-deterministic, countably-branching, cycle-free separated context for two probabilistic timed automata M_1 and M_2. Then there exists a distinguishing cycle-free separated context C' for M_1 and M_2 that at each state enabling discrete actions either enables two deterministic transitions or a unique probabilistic transition with two possible outcomes. C' is called a* time-deterministic, binary separated context.

Proof. The context C' is built from C in the same way as in the proof of Lemma 7.5.6. The constructions *shr* and *shf* work as well. The specific procedure is the same as the procedure followed in the proof of Lemma 11.5.3. ■

Lemma 11.5.8 *Let C be a distinguishing time-deterministic, binary separated context for two probabilistic timed automata M_1 and M_2. Then there exists a distinguishing time-deterministic, binary separated context C' for M_1 and M_2 where all the probabilistic transitions have a uniform distribution over two states. C' is called a* time-deterministic, balanced separated context.

Proof. The context C' is built from C in the same way as in the proof of Lemma 7.5.7. The specific procedure is the same as the procedure followed in the proof of Lemma 11.5.3. ■

Lemma 11.5.9 *Let C be a distinguishing time-deterministic, balanced separated context for two probabilistic timed automata M_1 and M_2. Then there exists a distinguishing time-deterministic, binary separated context C' for M_1 and M_2 with no internal actions and such that for each time t each discrete action appears exactly in one edge of the transition tree that leaves from a state whose time is t. C' is called a* time-deterministic, total balanced separated context.

Proof. The context C' is obtained from C by renaming all of its discrete actions so that for each time t each edge of the new transition relation leaving from a state whose current time is t has its own action. The proof of Lemma 7.5.8 applies. ■

Lemma 11.5.10 *Let C be a distinguishing time-deterministic, total balanced separated context for two probabilistic timed automata M_1 and M_2. Then there exists a distinguishing time-deterministic, total, cycle-free separated context C' for M_1 and M_2 that from every state enables one time-passage transition for each timed-action d, two deterministic transitions, and a probabilistic transition with a uniform distribution over two choices. C' is called a* complete context.

Proof. In this case it is enough to complete C by adding all the missing transitions and states. If $\mathcal{D}$ is a timed trace distribution of $M_1\|C$ that is not a timed trace distribution of $M_2\|C$, then it is enough to use on $M_1\|C'$ the same scheduler that is used in $M_1\|C$. In fact, since each new discrete transition of C' has a distinct action, none of the new discrete transitions of C' can be used in $M_2\|C'$ to generate $\mathcal{D}$, and since each state of C' is uniquely determined by the timed trace of all the executions leading to that state, none of the new time-passage transitions can be scheduled (this would affect the resulting timed trace distribution). ■

Lemma 11.5.11 *Let C be a distinguishing complete context for two probabilistic timed automata M_1 and M_2. Then the timed principal context is a distinguishing context for M_1 and M_2.*

Proof. The result is achieved in two steps. First the actions of C are renamed so that each state enables two deterministic transitions with actions *left* and *right*, a probabilistic transition with actions *pleft* and *pright*, and one transition for each time-passage action d. Call this context C_1. Then, by observing that the state of C_1 is uniquely determined by the timed trace of any timed execution leading to it, all the states of C_1 are collapsed into a unique one.

Thus, we need to show only that C_1 is a distinguishing context. The proof of Lemma 7.5.10 applies. ■

Lemma 11.5.12 *Let C_P be a distinguishing context for two probabilistic timed automata M_1 and M_2. Then the simple context C of Figure 7-6 augmented with a self-loop time-passage transition from state s_0 for each time-passage action d, where start is an action that does not appear in M_1 and M_2, is a distinguishing context for M_1 and M_2.*

Proof. The proof of Lemma 7.5.11 applies. ∎

Proof of Theorem 11.5.1. Let $M_1 \sqsubseteq_{DCt} M_2$. Then, from Lemma 11.5.12, $M_1\|C_P \sqsubseteq_{Dt} M_2\|C_P$. Conversely, let $M_1\|C_P \sqsubseteq_{Dt} M_2\|C_P$. Then, from Lemmas 11.5.3, 11.5.4, 11.5.5, 11.5.6, 11.5.7, 11.5.8, 11.5.9, 11.5.10, and 11.5.11, $M_1 \sqsubseteq_{DCt} M_2$. ∎

Chapter 12

Hierarchical Verification: Timed Simulations

12.1 Introduction

The simulation method extends to the timed framework almost directly. The main difference is that in a timed simulation that abstracts from internal computation we use moves (cf. Section 9.4) rather than weak combined transitions. The kind of results that we prove are a direct extension of similar results for the untimed model. In particular, probabilistic timed forward simulations are sound for the timed trace distribution precongruence.

12.2 Probabilistic Timed Simulations

We start directly with simulation relations that abstract from internal computation; the strong relations are essentially the same as for the untimed case.

For convenience assume that M_1 and M_2 do not have common states. A *probabilistic timed bisimulation* between two simple probabilistic timed automata M_1 and M_2 is an equivalence relation $\mathcal{R}$ over $states(M_1) \cup states(M_2)$ such that

1. each start state of M_1 is related to at least one start state of M_2, and vice versa;
2. for each pair of states $s_1 \;\mathcal{R}\; s_2$ and each transition $s_1 \xrightarrow{a} \mathcal{P}_1$ of either M_1 or M_2, there exists a move $s_2 \overset{a \upharpoonright ext(M_2)}{\rightsquigarrow} \mathcal{P}_2$ of either M_1 or M_2 such that $\mathcal{P}_1 \equiv_{\mathcal{R}} \mathcal{P}_2$.

We write $M_1 \simeq_{\mathrm{Pt}} M_2$ whenever $ext(M_1) = ext(M_2)$ and there is a probabilistic timed bisimulation between M_1 and M_2.

A *probabilistic timed simulation* between two simple probabilistic timed automata M_1 and M_2 is a relation $\mathcal{R} \subseteq states(M_1) \times states(M_2)$ such that

1. each start state of M_1 is related to at least one start state of M_2;
2. for each pair of states $s_1 \;\mathcal{R}\; s_2$ and each transition $s_1 \xrightarrow{a} \mathcal{P}_1$ of M_1, there exists a move $s_2 \overset{a \upharpoonright ext(M_2)}{\rightsquigarrow} \mathcal{P}_2$ of M_2 such that $\mathcal{P}_1 \sqsubseteq_{\mathcal{R}} \mathcal{P}_2$.

We write $M_1 \sqsubseteq_{\mathrm{Pt}} M_2$ whenever $ext(M_1) = ext(M_2)$ and there is a probabilistic timed simulation from M_1 to M_2. We denote the kernel of probabilistic timed simulation by $\equiv_{\mathrm{Pt}}$.

It is easy to check that $\simeq_{\mathrm{Pt}}$ is an equivalence relation, that $\sqsubseteq_{\mathrm{Pt}}$ is a preorder relation, and that both $\simeq_{\mathrm{Pt}}$ and $\sqsubseteq_{\mathrm{Pt}}$ are preserved by the parallel composition operator. It is also easy to verify that a weak probabilistic bisimulation is a probabilistic timed bisimulation and that a weak probabilistic simulation is a probabilistic timed bisimulation.

12.3 Probabilistic Timed Forward Simulations

A *probabilistic timed forward simulation* between two simple probabilistic timed automata M_1, M_2 is a relation $\mathcal{R} \subseteq$ *states*$(M_1) \times$ *Probs*(*states*(M_2)) such that

1. each start state of M_1 is related to at least one Dirac distribution over a start state of M_2;
2. for each $s \,\mathcal{R}\, \mathcal{P}'$, if $s \xrightarrow{a} \mathcal{P}_1$, then
   (a) for each $s' \in \Omega'$ there exists a probability space $\mathcal{P}_{s'}$ such that $s' \overset{a\lceil ext(M_2)}{\leadsto} \mathcal{P}_{s'}$, and
   (b) there exists a probability space $\mathcal{P}'_1$ of *Probs*(*Probs*(*states*(M_2))) satisfying $\mathcal{P}_1 \sqsubseteq_{\mathcal{R}} \mathcal{P}'_1$,

   such that $\sum_{s'\in\Omega'} P'[s']\mathcal{P}_{s'} = \sum_{\mathcal{P}\in\Omega'_1} P'_1[\mathcal{P}]\mathcal{P}$.

Denote the existence of a probabilistic timed forward simulation from M_1 to M_2 by $M_1 \sqsubseteq_{FSt} M_2$.

Proposition 12.3.1 *$\sqsubseteq_{FSt}$ is preserved by the parallel composition operator.*

Proof. Let $M_1 \sqsubseteq_{FSt} M_2$, and let $\mathcal{R}$ be a probabilistic timed forward simulation from M_1 to M_2. Let $\mathcal{R}'$ be a relation between *states*$(M_1) \times$ *states*(M_3) and *Probs*(*states*$(M_2) \times$ *states*(M_3)), defined as follows:

$$(s_1, s_3)\,\mathcal{R}'\,\mathcal{P} \text{ iff } \mathcal{P} = \mathcal{P}_2 \otimes \mathcal{D}(s_3) \text{ for some } \mathcal{P}_2 \text{ such that } s_1 \,\mathcal{R}\, \mathcal{P}_2.$$

The proof that $\mathcal{R}'$ satisfies Condition 1 and that Condition 2 is satisfied for each discrete transition of $M_1 \| M_2$ is essentially the proof of Proposition 8.5.1. Thus we need to show only that Condition 2 is satisfied by time-passage transitions.

Let $(s_1, s_3)\,\mathcal{R}'\,\mathcal{P}_2 \otimes \mathcal{D}(s_3)$, and let $(s_1, s_3) \xrightarrow{d} (s'_1, s'_3)$, where $s_1 \xrightarrow{d} s'_1$, and $s_3 \xrightarrow{d} s'_3$. From the definition of a probabilistic timed forward simulation, for each $s \in \Omega_2$ there exists a move $s_2 \overset{d}{\leadsto} \mathcal{P}_s$ of M_2, and there exists a probability space $\mathcal{P}'_2$ of *Probs*(*Probs*(*states*(M_2))), such that

$$\sum_{s\in\Omega_2} P_2[s]\mathcal{P}_s = \sum_{\mathcal{P}\in\Omega'_2} P'_2[\mathcal{P}]\mathcal{P}, \tag{12.1}$$

and

$$\mathcal{D}(s'_1) \sqsubseteq_{\mathcal{R}} \mathcal{P}'_2. \tag{12.2}$$

Moreover, from the definition of a probabilistic timed automaton, there is a trajectory ω_3 for $s_3 \xrightarrow{d} s'_3$.

For each $s \in \Omega_2$, let $\mathcal{O}_s$ be a generator for $s \overset{d}{\leadsto} \mathcal{P}_s$. Define a new generator $\mathcal{O}'_s$ as follows: for each finite execution fragment α of $M_2 \| M_3$ starting in (s, s_3),

1. if $\mathcal{O}_s(\alpha\lceil M_2) = (s', \mathcal{P})$, where $(s', \mathcal{P}) = \sum_i p_i(s', a_i, \mathcal{P}_i)$, each $(s', a_i, \mathcal{P}_i)$ is a transition of M_2, and $\alpha\lceil M_3$ is consistent with ω_3, i.e., for each prefix α' of α, $\mathit{lstate}(\alpha')\lceil M_3 = \omega_3(\mathit{ltime}(\alpha'))$, then letting s_3'' denote $\mathit{lstate}(\alpha\lceil M_3)$,

   $$\mathcal{O}'_s(\alpha) = \sum_i p_i((s', s_3''), a_i, \mathcal{P}_i \otimes \mathcal{P}'_i),$$

   where $\mathcal{P}'_i = \mathcal{D}(s_3'')$ if a_i is a discrete action, and $\mathcal{P}'_i = \mathcal{D}(\omega_3(\mathit{ltime}(\alpha) + a_i))$ if a_i is a time-passage action.

2. otherwise, $\mathcal{O}'_s(\alpha) = \mathcal{D}(\delta)$.

The move generated by each $\mathcal{O}'_s$ is $(s, s_3) \overset{d}{\rightsquigarrow} \mathcal{P}_s \otimes \mathcal{D}(s_3')$. In fact, an execution fragment α of $M_2\|M_3$ is terminal for $\mathcal{O}'_s$ iff $\alpha\lceil M_2$ is terminal for $\mathcal{O}_s$ and $\mathit{lstate}(\alpha\lceil M_3) = s_3'$, and thus $\Omega_{\mathcal{O}'_s} = \Omega_s \times \mathcal{D}(s_3')$. Moreover, for each $\alpha \in \Omega_{\mathcal{O}'_s}$, $P_\alpha^{\mathcal{O}'_s} = P_{\alpha\lceil M_2}^{\mathcal{O}_s}$.

Denote $\mathcal{P}_s \otimes \mathcal{D}(s_3')$ by $\mathcal{P}_{(s,s_3)}$. Then, for each $(s, s_3) \in \Omega_2 \otimes \mathcal{D}(s_3)$, we have identified a move $(s, s_3) \overset{a}{\rightsquigarrow} \mathcal{P}_{(s,s_3)}$. These are the spaces of Condition 2.a in the definition of a probabilistic timed forward simulation.

From this point the proof proceeds exactly in the same way as the proof of Proposition 8.5.1. No modification of the text is necessary. ■

12.4 The Execution Correspondence Theorem: Timed Version

The execution correspondence theorem of Chapter 8 extends easily to the timed framework. In this section we define the notion of a timed execution correspondence structure, show the timed version of the execution correspondence theorem, and, as a consequence, show that probabilistic timed forward simulations are transitive.

The timed execution correspondence theorem is stated in terms of the probabilistic executions of a probabilistic timed automaton; however, it is easy to see that the same result can be extended to probabilistic timed executions: the execution correspondence theorem talks about countably many states of a probabilistic timed execution; all the other points can be described by arbitrary trajectories.

12.4.1 Timed Execution Correspondence Structure

The definition of a fringe for a probabilistic timed execution is the same as the definition of a fringe for a probabilistic execution. For the definition of $\mathit{fringe}(H, i)$ the only difference is in the way the length of a state of H is measured, and thus the definition given for probabilistic automata is still valid.

Let $\mathcal{R}$ be a probabilistic timed forward simulation from M_1 to M_2. A *timed execution correspondence structure* via $\mathcal{R}$ is a tuple (H_1, H_2, m, S), where H_1 is a probabilistic execution of M_1, H_2 is a probabilistic execution of M_2, m is a mapping from natural numbers to fringes of M_2, and S is a mapping from natural numbers to probability distributions of $\mathit{Probs}(\mathit{Probs}(\mathit{states}(H_2)))$, such that

1. For each i, $m(i) \leq m(i+1)$;
2. For each state q_2 of H_2, $\lim_{i\to\infty} \sum_{q\in\Omega_i | q_2 \leq q} P_i[q] = P_H[C_q]$;
3. Let $q_1 \mathcal{R} (\Omega, \mathcal{F}, P)$ iff for each $q \in \Omega$, *t-trace*$(q) =$ *t-trace*(q_1), and either

   (a) q_1 does not end in δ, each state of Ω does not end in δ, and *lstate*$(q_1) \mathcal{R}$ *lstate*$(\mathcal{P})$, or

   (b) q_1 and each state of Ω end in δ and *lstate*$(\delta$-*strip*$(q_1)) \mathcal{R}$ *lstate*$(\delta$-*strip*$(\mathcal{P}))$.

   Then, for each $i \geq 0$, $m(i) = \sum_{\mathcal{P}\in\Omega_{S(i)}} P_{S(i)}[\mathcal{P}]\mathcal{P}$, and *fringe*$(H_1, i) \sqsubseteq_{\mathcal{R}} S(i)$.
4. Let, for each $i \geq 0$, each $q_1 \in$ *fringe*(H_1, i), and each $q_2 \in$ *states*(H_2), $W_i(q_1, q_2) \triangleq \sum_{\mathcal{P}} w_i(q_1, \mathcal{P}) P[q_2]$. If $W_i(q_1, q_2') = 0$ for each prefix or extension q_2' of q_2, then, for each extension q_1' of q_1 such that $q_1' \in$ *fringe*$(H_1, i+1)$, and each prefix or extension q_2' of q_2, $W_{i+1}(q_1', q_2') = 0$.

12.4.2 The Main Theorem

Theorem 12.4.1 *Let $M_1 \sqsubseteq_{FS} M_2$ via the probabilistic timed forward simulation $\mathcal{R}$, and let H_1 be a probabilistic execution of M_1. Then there exists a probabilistic execution H_2 of M_2, a mapping m from natural numbers to fringes of M_2, and a mapping S from natural numbers to probability distributions of Probs(Probs(states(H_2))), such that (H_1, H_2, m, S) is an execution correspondence structure via $\mathcal{R}$.*

Proof. The proof has exactly the same structure as the proof of Theorem 8.6.1. Note that the only difference between this theorem and Theorem 8.6.1 is in Condition 3, where we use timed traces rather than traces. ■

12.4.3 Transitivity of Probabilistic Timed Forward Simulations

The timed execution correspondence theorem can be used to show that probabilistic timed forward simulations are transitive, i.e., if $M_1 \sqsubseteq_{FSt} M_2$ and $M_2 \sqsubseteq_{FSt} M_3$, then $M_1 \sqsubseteq_{FSt} M_3$. The proof of this result follows the same lines as the corresponding proof in the untimed case (cf. Section 8.6.4), where combined transitions are replaced by moves and traces are replaced by timed traces. We leave the details of the proof to the reader.

12.5 Soundness for Timed Trace Distributions

As for the untimed model, the timed execution correspondence theorem can be used to show that probabilistic timed forward simulations are sound for the timed trace distribution precongruence. Since $\sqsubseteq_{FSt}$ is a precongruence, it is enough to show that $\sqsubseteq_{FSt}$ is sound for the timed trace distribution preorder.

Proposition 12.5.1 *If $M_1 \sqsubseteq_{FSt} M_2$, then $M_1 \sqsubseteq_{Dt} M_2$.*

Proof. Let $M_1 \sqsubseteq_{FSt} M_2$, and let H_1 be a probabilistic execution of M_1 that leads to a timed trace distribution $\mathcal{D}_1$. From Lemma 12.4.1, there exists a probabilistic execution H_2 of M_2 that corresponds to H_1 via some mappings m, S. We show that H_2 leads to a timed trace distribution $\mathcal{D}_2$ that is equivalent to $\mathcal{D}_1$.

Consider a cone C_β of $\mathcal{D}_1$. The cone C_β can be expressed as a union of cones of $\mathcal{P}_{H_1}$, and thus its measure can be expressed as

$$\lim_{i \to \infty} \sum_{q_1 \in fringe(H_1,i) | \beta \leq t\text{-}trace(q_1)} P_{H_1}[C_{q_1}]. \tag{12.3}$$

Consider a cone C_β of $\mathcal{D}_2$. The cone C_β can be expressed as a union of cones of $\mathcal{P}_{H_2}$, and thus its measure can be expressed as

$$\lim_{i \to \infty} \sum_{q_2 \in m(i) | \beta \leq t\text{-}trace(q_2)} P_{m(i)}[q_2]. \tag{12.4}$$

The reason for Expression (12.4) is that at the limit each cone expressing the occurrence of β is captured completely.

Thus, it is sufficient to show that for each finite β and each i,

$$\sum_{q_1 \in fringe(H_1,i) | \beta \leq t\text{-}trace(q_1)} P_{H_1}[C_{q_1}] = \sum_{q_2 \in m(i) | \beta \leq t\text{-}trace(q_2)} P_{m(i)}[q]. \tag{12.5}$$

From this point the proof proceeds exactly as the proof of Proposition 8.7.1. ∎

Chapter 13

Conclusion

13.1 Have we Met the Challenge?

We have developed a model for the description of randomized distributed real-time systems, and we have investigated how the new model can be used for the analysis of algorithms. The main idea behind the model is to extend labeled transition systems to account for randomization in such a way that probabilistic behavior and nondeterministic behavior are clearly distinct.

We have shown how commonly used informal statements can be formulated in the new formalism, and we have shown how such statements can be proved to be correct in a formal and rigorous way. In particular, we have developed verification techniques that resemble the common ways in which randomized algorithms are analyzed. The main improvement is that now we have a collection of results that allow us to determine when a specific argument can be used safely. Furthermore, we have shown how to derive upper bounds to the complexity of a randomized distributed algorithm using an ordinary time complexity measure as well as more abstract complexity measures like "number of rounds in an asynchronous computation".

Finally, we have extended several verification techniques that are commonly used within the labeled transition system model. We have extended the trace semantics of labeled transition systems and several of the existing simulation relations for labeled transition systems. In particular, all our preorder relations are compositional and the simulation relations are sound for the trace-based semantics. Although we have not presented any example of verification using simulations, except for two toy examples based on coin flips, we are confident that in the future the method based on simulations will become of practical relevance as it happened for ordinary automata.

Therefore, we can claim that we have met the challenge given by randomization at least partially. Surely we understand much more of the problem than before. The fact that we have been able to prove new results about randomized algorithms is a positive sign. In particular, Aggarwal [Agg94] used successfully the technique presented in this thesis for the verification of the randomized self-stabilizing algorithm of Aggarwal and Kutten [AK93], which is not trivial at all; during the verification process Aggarwal discovered also a subtle bug in the original protocol. In the measure in which the power of a proof method is evaluated based on the bugs that such method helps to discover, our methodology has achieved something. Indeed we have discovered another bug on one existing algorithm, and the main issue is that we did not have to work much to discover such a bug; essentially it was sufficient to try to reformulate the proof of correctness in our framework.

13.2 The Challenge Continues

Although we have improved considerably our understanding of randomization in distributed computation, what we have discovered looks like the tip of the iceberg. We have addressed several problems, and in solving them we have addressed more the basic methodology rather than an extensive analysis of all the possible solutions. Therefore, there are several directions for further research that can be pursued. Here we suggest some of the most important directions.

13.2.1 Discrete versus Continuous Distributions

Throughout this thesis we have assumed that the probability distributions associated with the transitions of a probabilistic automaton are discrete. Although such assumption is sufficiently general for the study of several randomized algorithms, several other real-time systems are better described by using continuous distributions. Examples involve algorithms for transmission of data along a common wire, scheduling algorithms for massively parallel machines, and queuing systems. Moreover, continuous distributions would be more suitable for the study of randomized hybrid systems.

The extension of the theory to continuous distributions involves nontrivial measure theoretical problems. In particular it is not the case any more that any union of cones is measurable; thus, not even the event that expresses the occurrence of an action or the reachability of a state is measurable in general. The events with probability 0 need a more careful treatment within the model with continuous distributions. It is likely that some restrictions must be imposed to the model to ensure that some minimal set of events is measurable. Examples of restricted models with continuous distributions are the automata of Alur, Courcuobetis and Dill [ACD91a, ACD91b], where the time that elapses between two transitions is governed by an exponential distribution or by a distribution which is non zero in a finite collection of closed intervals, and the models of [GHR93, Hil93, BDG94], where the time between the occurrence of two actions is assumed to be distributed exponentially. Exponential distributions occur in several real systems and are easy to model due to their memoryless structure. However, other distributions should be studied.

13.2.2 Simplified Models

Within the context of ordinary automata Lynch and Tuttle [LT87] have developed a model of I/O automata. The model enforces a distinction between Input actions and Output actions within an automaton, and requires that input actions are enabled from every state. Furthermore, in a parallel composition context each action is required to be the output or internal action of at most one process, i.e., each action is under the control of at most one process. Based on the Input/Output distinction Lynch and Tuttle can introduce fairness in the model in a natural way, and in particular they can use the trace semantics as a meaningful notion of implementation. In general the trace semantics is not meaningful as a notion of implementation since, for example, it is not sensitive to deadlock. The advantage of the use of traces is that traces are easy to deal with.

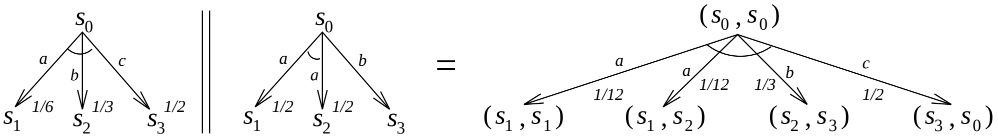


Figure 13-1: Synchronization for probabilistic I/O automata.

For this reason, it makes sense to study a theory of probabilistic I/O automata as an extension of the model of [LT87] and as a restriction of our model. An interesting point of a model with I/O distinction is that it is possible to relax the requirement that all the transitions of a probabilistic I/O automaton are simple. In particular, only the transitions with input actions need to be simple, while all the others can be general. The parallel composition can be defined easily since a non-simple transition synchronizes only with simple transitions. Figure 13-1 gives an example of synchronization between a transition with three output actions a, b, c and two transitions of an I/O automaton with just two input actions a, b. A similar observation was made also by Wu, Stark and Smolka in [WSS94].

A restricted timed model with I/O distinction is introduced by Merrit, Modugno and Tuttle [MMT91]. In particular timing constraints can be described only by giving upper and lower bounds to the time it takes for a process to perform the next transition whenever it is ready to do so. MMT automata turned out to be sufficient for the modeling of several distributed systems, and in particular, due to their simple structure, made the analysis simpler than by using the full automaton model. Once again, a study of the probabilistic version of the MMT model would be useful. The proofs that we have illustrated in Chapter 12 could be carried out in the probabilistic MMT model as well.

Finally, the analysis of a system can be simplified by studying time-deterministic probabilistic timed automata, i.e., probabilistic timed automata such that from each state s and each time d there is at most one state reachable from s in time d. In fact, if a system is time-deterministic, then the end points of a time-passage transition determine completely the trajectory that is spanned. Therefore, trajectories could be removed also from the direct analysis of randomized timed algorithms. It turns out that most of the times an algorithm can be described as a time-deterministic probabilistic automaton. Probabilistic MMT automata are an example of time-deterministic probabilistic automata.

13.2.3 Beyond Simple Probabilistic Automata

The study of parallel composition and of the simulation relations of this thesis is done within the context of simple probabilistic automata. The main problem is that we did not find any reasonable definition of parallel composition for general probabilistic automata that is consistent with our synchronization style. We have just observed that in the presence of an Input/Output distinction it is possible to relax the simplicity condition and yet obtain a meaningful notion of parallel composition. It would be interesting to investigate other mechanisms that give a meaning to general probabilistic automata and yet work as we expect in the simple case.

13.2.4 Completeness of the Simulation Method

We have provided several simulation and bisimulation relations for probabilistic automata and probabilistic timed automata, and we have shown that they are sound for the trace distribution precongruence and the timed trace distribution precongruence, respectively. However, we have not shown any completeness result for probabilistic forward simulations and probabilistic forward timed simulations. In [LV93a, LV95] it is shown that forward simulations together with another kind of simulations called *backward simulations* are sound and complete for the trace preorder. Are probabilistic forward simulations complete for the trace distribution preorder? If not, is there an equivalent of backward simulations that can lead to completeness?

13.2.5 Testing Probabilistic Automata

We have presented the trace distribution semantics as an example of a semantics based on abstract observations. Another widely known semantics for ordinary automata is the failure semantics of Brookes, Hoare and Roscoe [BHR84], which in turn is connected to the testing preorders of De Nicola and Hennessy [DH84]. Similarly to the trace distribution semantics, it should be possible to extend the failure semantics to the probabilistic framework and find a sufficiently powerful context to distinguish probabilistic automata that are not in the corresponding precongruence relation. Possibly, a related theory of testing in the style of [DH84] should be defined. It is very likely that the new testing preorders will be similar to those of Yi and Larsen [YL92]. Other theories of testing for probabilistic automata are studied in [Chr90b, Chr90a, CSZ92, YCDS94] and are explained in Section 2.2.

13.2.6 Liveness in Probabilistic Automata

In the extension of the notion of an execution of an automaton we have obtained a parallelism between the theory of ordinary automata and the theory of probabilistic automata. In this parallelism also the notion of liveness has found its place, although we have not addressed the issue in this thesis. In ongoing research we have given a simple definition of a live probabilistic automaton as a pair (M, L) where L is an arbitrary subset of the probabilistic executions of M, and we have shown that the *live trace distribution precongruence* can be defined easily and can be characterized by a *live principal context*, which is essentially the principal context paired with the set of its probabilistic executions. However, lot of work remains to be done within the theory of liveness.

First of all it would be useful to study how the definition of safety and liveness properties of Alpern and Schneider [AS85] extends to the probabilistic framework and what consequences such extension has. Furthermore, the use of the live trace preorder within ordinary automata makes sense as a notion of implementation in the presence of I/O distinction and of a property called *receptiveness* or *environment-freedom* [Dil88, AL93, GSSL94]. It would be useful to study the theory of receptiveness of [Dil88, AL93] and of environment-freedom of [GSSL94] in the context of randomization. In this case, differently from [GSSL94], the environment is expressed by a function rather than by a sequence of actions. However, non-trivial problems arise in imposing restrictions to the behavior of the environment.

13.2.7 Temporal Logics for Probabilistic Systems

In the chapters on direct analysis we have identified a collection of probabilistic statements that are useful for the analysis of algorithms. However, there are several other statements that can be of interest. It would be desirable to find a probabilistic temporal logic that expresses as many properties as possible. The probabilistic modal logic of [LS89] is a direct extension of the modal logic of Hennessy and Milner [HM85] for reactive processes, but it is not sufficiently powerful to deal with nondeterminism; similarly, the extended probabilistic logic of [LS92] is not sufficiently powerful. The Probabilistic Computation Tree Logic of [HJ89, Han94] captures more the consequences of the interplay between probability and nondeterminism; in [SL94] PCTL is generalized also to probabilistic systems with internal actions (WPCTL). However, there are still properties that are useful and do not seem to be expressible in WPCTL. Specifically, we do not know how to express a property of the kind "after something has happened, no matter where I am, something else will happen with probability at least p". Is there something missing in WPCTL? What would be a more appropriate temporal logic?

Another issue is the relationship between the simulation method and temporal logic. That is, if a probabilistic automaton implements another probabilistic automaton according to some implementation relation (e.g., trace distribution precongruence, probabilistic simulation, probabilistic forward simulation, etc.), what can we say about the implementation? What properties of the specification are satisfied by the implementation? More generally, given a probabilistic temporal logic and a preorder relation, what fragment of the logic is preserved by the preorder relation? Somehow it is implicit that whenever we use some preorder relation as a notion of implementation we are interested only in the properties that are preserved by such relation; however, we need to know what are those properties. In [SL95] we have stated that weak probabilistic simulation preserve a large fragment of WPCTL and that weak probabilistic bisimulations preserve WPCTL. The results of [SL95] can be proved easily given the results of this thesis. However, more work in this direction is necessary. In particular, some completeness results would be useful.

13.2.8 More Algorithms to Verify

In this thesis we have illustrated our direct verification technique by proving the correctness of the randomized dining philosophers algorithm of Lehmann and Rabin [LR81] and of the randomized agreement protocol of Ben-Or [BO83]. In [Agg94] Aggarwal uses our model to verify the correctness of the self-stabilizing minimum weight spanning tree randomized algorithm of Aggarwal and Kutten [AK93]. However, the technique should be tested against many other algorithms. We are currently investigating the agreement protocol of Aspnes and Herlihy [AH90] and the randomized mutual exclusion algorithm of Pnueli and Zuck [PZ86]. Based on the little experience that we have gained, we can say that the model provides us with a systematic way of analyzing those algorithms, and in particular it provides us with a simple methodology to identify the critical points of an algorithm.

It is very likely that new coin lemmas need to be developed together with other techniques for the actual computation of the probability of an event. A technique that needs further development is the partition technique of Section 6.7. The analysis of other algorithms should make clear what other techniques are necessary. Also, playing with the toy resource allocation protocol of Chapter 5 can be very instructive. Although the protocol is simple, its analysis highlights several of the issues that arise in randomized distributed computation.

It is also plausible, as it happened for non-probabilistic distributed algorithms, that some complex protocols can be verified more easily by using the simulation method. Finding those algorithms would be an optimal way to test the hierarchical verification method and possibly to improve it.

13.2.9 Automatic Verification of Randomized Systems

Formal verification usually involves two levels of analysis. First, an algorithm is analyzed at a high level by using the intuition that designers have of their own algorithm; then, a more detailed verification of the high level claims is carried out in order to guarantee correctness. The low level analysis is very tedious and involves checking a whole lot of uninteresting details. On the other hand, several times the low level analysis is the only way to discover flaws in the intuitions about an algorithm.

Fortunately, the low level analysis is amenable to automatic verification, although the research in this area is still in progress. Model checking [EC82, CES83] is certainly a useful technique; in [SGG$^+$93] it is shown how a theorem prover can be used to help in the verification of a protocol using simulations; in [PS95] we have investigated how a randomized algorithm can be verified mechanically once the high level proof is formulated. However, there is still a lot of work that needs to be done. It would be interesting to study how model checking and theorem proving could be integrated to automatize part of the verification of an algorithm.

13.3 The Conclusion's Conclusion

To say what we have done in one sentence, we have provided a new way of reasoning about randomized systems that integrates both the theoretical aspects of modeling and the basic requirements for usage in practice. From the modeling point of view we have distinguished between nondeterminism and probability explicitly and we have extended the main semantics that are available within the labeled transition systems model; from the point of view of verification we have formalized some of the common informal arguments about randomized algorithms and we have provided guidelines to determine whether an argument can be used safely. Furthermore, we have provided a systematic way to analyze the complexity of randomized algorithms. All our results are compatible with previous work.

As we have seen in the previous section, there are still many open problems in this area. Here we hope to have stimulated the curiosity of the reader to go much further. Needless to say that for us (me) working on this project was a continuous discovery.

Bibliography

[ACD91a] R. Alur, C. Courcoubetis, and D.L. Dill. Model-checking for probabilistic real-time systems. In J. Leach Albert, B. Monien, and M. Rodríguez, editors, *Proceedings 18^{th} ICALP,* Madrid, volume 510 of *Lecture Notes in Computer Science*, pages 115–136. Springer-Verlag, 1991.

[ACD91b] R. Alur, C. Courcoubetis, and D.L. Dill. Verifying automata specifications of probabilistic real-time systems. In de Bakker et al. [dBHRR91], pages 28–44.

[ACS94] B. Awerbuch, L. Cowen, and M.A. Smith. Efficient asynchronous distributed symmetry breaking. In *Proceedings of the 26^{th} Annual ACM Symposium on Theory of Computing*, 1994.

[Agg94] S. Aggarwal. Time optimal self-stabilizing spanning tree algorithms. Technical Report MIT/LCS/TR-632, MIT Laboratory for Computer Science, 1994. Master's thesis.

[AH90] J. Aspnes and M.P. Herlihy. Fast randomized consensus using shared memory. *Journal of Algorithms*, 15(1):441–460, September 1990.

[AK93] S. Aggarwal and S. Kutten. Time optimal self stabilizing spanning tree algorithms. In R.K. Shyamasundar, editor, *13th International Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 761 of *Lecture Notes in Computer Science*, pages 400–410, Bombay, India., December 1993. Springer-Verlag.

[AL91] M. Abadi and L. Lamport. An old-fashioned recipe for real time. In de Bakker et al. [dBHRR91], pages 1–27.

[AL93] M. Abadi and L. Lamport. Composing specifications. *ACM Transactions on Programming Languages and Systems*, 15(1):73–132, 1993.

[AS85] B. Alpern and F.B. Schneider. Defining liveness. *Information Processing Letters*, 21(4):181–185, 1985.

[BBS92] J.C.M. Baeten, J.A. Bergstra, and S.A. Smolka. Axiomatizing probabilistic processes: ACP with generative probabilities. In Cleaveland [Cle92], pages 472–485.

[BDG94] M. Bernardo, L. Donatiello, and R. Gorrieri. Modeling and analyzing concurrent systems with MPA. In U. Herzog and M. Rettelbach, editors, *Proceedings of* *the Second Workshop on Process Algebras and Performance Modelling* (PAPM), Erlangen, Germany, pages 175–189, 1994.

[BFJ+82] J. Burns, M. Fisher, P. Jackson, N.A. Lynch, and G. Peterson. Data requirements for implementation of n-process mutual exclusion using a single shared variable. *Journal of the ACM*, 29(1):183–205, 1982.

[BG91] J.C.M. Baeten and J.F. Groote, editors. *Proceedings of CONCUR 91,* Amsterdam, volume 527 of *Lecture Notes in Computer Science.* Springer-Verlag, 1991.

[BHR84] S.D. Brookes, C.A.R. Hoare, and A.W. Roscoe. A theory of communicating sequential processes. *Journal of the ACM*, 31(3):560–599, 1984.

[BK90] J.C.M. Baeten and J.W. Klop, editors. *Proceedings of CONCUR 90,* Amsterdam, volume 458 of *Lecture Notes in Computer Science.* Springer-Verlag, 1990.

[BM89] B. Bloom and A. Meyer. A remark on bisimulation between probabilistic processes. In *Proceedings of the Symposium on Logical Foundations of Computer Science*, volume 363 of *Lecture Notes in Computer Science*, pages 26–40, 1989.

[BO83] M. Ben-Or. Another advantage of free choice: completely asynchronous agreement protocols. In *Proceedings of the* 2^{nd} *Annual ACM Symposium on Principles of Distributed Computing,* Montreal, Quebec, Canada, August 1983.

[BPV94] D. Bosscher, I. Polak, and F. Vaandrager. Verification of an audio control protocol. Technical Report CS-R9445, CWI, Amsterdam, July 1994.

[BW90] J.C.M. Baeten and W.P Weijland. *Process Algebra.* Cambridge Tracts in Theoretical Computer Science 18, Cambridge University Press, 1990.

[CES83] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2), 1983.

[Chr90a] I. Christoff. Testing equivalences and fully abstract models for probabilistic processes. In Baeten and Klop [BK90], pages 126–140.

[Chr90b] I. Christoff. *Testing Equivalences for Probabilistic Processes.* PhD thesis, Department of Computer Science, Uppsala University, 1990.

[Chr93] L. Christoff. *Specification and Verification Methods for Probabilistic Processes.* PhD thesis, Department of Computer Science, Uppsala University, 1993.

[Cle92] W.R. Cleaveland, editor. *Proceedings of CONCUR 92,* Stony Brook, NY, USA, volume 630 of *Lecture Notes in Computer Science.* Springer-Verlag, 1992.

[CM88] K.M. Chandi and J. Misra. *Parallel Program Design: A Foundation.* Addison-Wesley, 1988.

[CSZ92] R. Cleaveland, S.A. Smolka, and A. Zwarico. Testing preorders for probabilistic processes (extended abstract). In *Proceedings* 19^{th} *ICALP,* Madrid, volume 623 of *Lecture Notes in Computer Science*, pages 708–719. Springer-Verlag, 1992.

[CY88] C. Courcoubetis and M. Yannakakis. Verifying temporal properties of finite-state probabilistic programs. In 29^{th} *Annual Symposium on Foundations of Computer Science*, pages 338–345, 1988.

[CY90] C. Courcoubetis and M. Yannakakis. Markov decision procedures and regular events. In M. Paterson, editor, *Proceedings* 17^{th} *ICALP,* Warwick, volume 443 of *Lecture Notes in Computer Science*, pages 336–349. Springer-Verlag, July 1990.

[dBHRR91] J.W. de Bakker, C. Huizing, W.P. de Roever, and G. Rozenberg, editors. *Proceedings of the REX Workshop "Real-Time: Theory in Practice"*, volume 600 of *Lecture Notes in Computer Science*. Springer-Verlag, 1991.

[DeN87] R. De Nicola. Extensional equivalences for transition systems. *Acta Informatica*, 24:211–237, 1987.

[DH84] R. De Nicola and M. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.

[Dil88] D. Dill. *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits.* ACM Distinguished Dissertations. MIT Press, 1988.

[EC82] E.A. Emerson and E.C. Clarke. Using branching time temporal logic to synthesize synchronous skeletons. *Science of Computer Programming*, 2:241–266, 1982.

[FLP85] M. Fischer, N. Lynch, and M. Paterson. Impossibility of distributed consensus with a family of faulty process. *Journal of the ACM*, 32(2):374–382, April 1985.

[GHR93] N. Götz, U. Herzog, and M. Rettelbach. Multiprocessor and distributed system design: the integration of functional specification and performance analysis using stochastic process algebras. In L. Donatiello and R. Nelson, editors, *Performance Evaluation of Computer and Communication Systems. Joint Tutorial Papers of Performance '93 and Sigmetrics '93*, volume 729 of *Lecture Notes in Computer Science*, pages 121–146. Springer-Verlag, 1993.

[GJS90] A. Giacalone, C.C Jou, and S.A. Smolka. Algebraic reasoning for probabilistic concurrent systems. In *Proceedings of the Working Conference on Programming Concepts and Methods (IFIP TC2),* Sea of Galilee, Israel, 1990.

[Gla90] R.J. van Glabbeek. The linear time – branching time spectrum. In Baeten and Klop [BK90], pages 278–297.

[Gla93] R.J. van Glabbeek. The linear time – branching time spectrum ii. The semantics of sequential systems with silent moves. In E. Best, editor, *Proceedings of CONCUR 93,* Hildesheim, Germany, volume 715 of *Lecture Notes in Computer Science*, pages 66–81. Springer-Verlag, 1993.

[GSB94] R. Gupta, S.A. Smolka, and S. Bhaskar. On randomization in sequential and distributed algorithms. *ACM Computing Surveys*, 26(1):1–86, 1994.

[GSSL94] R. Gawlick, R. Segala, J.F. Søgaard-Andersen, and N.A. Lynch. Liveness in timed and untimed systems. In S. Abiteboul and E. Shamir, editors, *Proceedings* 21^{th} *ICALP,* Jerusalem, volume 820 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994. A full version appears as MIT Technical Report number MIT/LCS/TR-587.

[GSST90] R.J. van Glabbeek, S.A. Smolka, B. Steffen, and C.M.N. Tofts. Reactive, generative, and stratified models of probabilistic processes. In *Proceedings* 5^{th} *Annual Symposium on Logic in Computer Science,* Philadelphia, USA, pages 130–141. IEEE Computer Society Press, 1990.

[Hal50] P.R. Halmos. *Measure Theory.* Springer-Verlag, 1950.

[Han91] H. Hansson. *Time and Probability in Formal Design of Distributed Systems.* PhD thesis, Department of Computer Science, Uppsala University, 1991.

[Han94] H. Hansson. *Time and Probability in Formal Design of Distributed Systems*, volume 1 of *Real-Time Safety Critical Systems.* Elsevier, 1994.

[Hil93] J. Hillston. PEPA: Performance enhanced process algebra. Technical Report CSR-24-93, Department of Computer Science, University of Edimburgh (UK), 1993.

[Hil94] J. Hillston. *A Compositional Approach to Performance Modeling.* PhD thesis, Department of Computer Science, University of Edimburgh (UK), 1994.

[HJ89] H. Hansson and B. Jonsson. A framework for reasoning about time and reliability. In *Proceedings of the* $10^{\rm th}$ *IEEE Symposium on Real-Time Systems,* Santa Monica, Ca., 1989.

[HJ90] H. Hansson and B. Jonsson. A calculus for communicating systems with time and probabilities. In *Proceedings of the* $11^{\rm th}$ *IEEE Symposium on Real-Time Systems,* Orlando, Fl., 1990.

[HM85] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32(1):137–161, 1985.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall International, Englewood Cliffs, 1985.

[HS85] S. Hart and M. Sharir. How to schedule if you must. *SIAM Journal on Computing*, 14:991–1012, 1985.

[HSP83] S. Hart, M. Sharir, and A. Pnueli. Termination of probabilistic concurrent programs. *ACM Transactions on Programming Languages and Systems*, 5(3):356–380, 1983.

[JHY94] B. Jonsson, C. Ho-Stuart, and W. Yi. Testing and refinement for nondeterministic and probabilistic processes. In Langmaack, de Roever, and Vytopil, editors, *Proceedings of the Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 863 of *Lecture Notes in Computer Science*, pages 418–430, 1994.

[JL91] B. Jonsson and K.G. Larsen. Specification and refinement of probabilistic processes. In *Proceedings of the 6th IEEE Symposium on Logic in Computer Science*, pages 266–277, Amsterdam, July 1991.

[Jon91] B. Jonsson. Simulations between specifications of distributed systems. In Baeten and Groote [BG91], pages 346–360.

[JP89] C. Jones and G. Plotkin. A probabilistic powerdomain of evaluations. In *Proceedings* 4^{th} *Annual Symposium on Logic in Computer Science,* Asilomar, California, pages 186–195. IEEE Computer Society Press, 1989.

[JP94] B. Jonsson and J. Parrow, editors. *Proceedings of CONCUR 94,* Uppsala, Sweden, volume 836 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994.

[JS90] C.C. Jou and S.A. Smolka. Equivalences, congruences, and complete axiomatizations for probabilistic processes. In Baeten and Klop [BK90], pages 367–383.

[JY95] B. Jonsson and W. Yi. Compositional testing preorders for probabilistic processes. In *Proceedings* 10^{th} *Annual Symposium on Logic in Computer Science,* San Diego, California. IEEE Computer Society Press, 1995.

[Kar90] R.M. Karp. An introduction to randomized algorithms. Technical Report TR-90-024, Computer Science Division, University of California, Berkeley, CA, 1990.

[Kel76] R. Keller. Formal verification of parallel programs. *Communications of the ACM*, 7(19):561–572, 1976.

[KR92] E. Kushilevitz and M. Rabin. Randomized mutual exclusion algorithms revisited. In *Proceedings of the* 11^{th} *Annual ACM Symposium on Principles of Distributed Computing,* Quebec, Canada, pages 275–284, 1992.

[LR81] D. Lehmann and M. Rabin. On the advantage of free choice: a symmetric and fully distributed solution to the dining philosophers problem. In *Proceedings of the 8*$^{\rm th}$ *Annual ACM Symposium on Principles of Programming Languages*, pages 133–138, January 1981.

[LS82] D. Lehmann and S. Shelah. Reasoning with time and chance. *Information and Control*, 53:165–198, 1982.

[LS89] K.G. Larsen and A. Skou. Bisimulation through probabilistic testing. In *Conference Record of the* 16^{th} *ACM Symposium on Principles of Programming Languages,* Austin, Texas, pages 344–352, 1989.

[LS91] K.G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, September 1991.

[LS92] K.G. Larsen and A. Skou. Compositional verification of probabilistic processes. In Cleaveland [Cle92], pages 456–471.

[LSS94] N.A. Lynch, I. Saias, and R. Segala. Proving time bounds for randomized distributed algorithms. In *Proceedings of the* 13^{th} *Annual ACM Symposium on Principles of Distributed Computing,* Los Angeles, CA, pages 314–323, 1994.

[LT87] N.A. Lynch and M.R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proceedings of the* 6^{th} *Annual ACM Symposium on Principles of Distributed Computing*, pages 137–151, Vancouver, Canada, August 1987. A full version is available as MIT Technical Report MIT/LCS/TR-387.

[LV91] N.A. Lynch and F.W. Vaandrager. Forward and backward simulations for timing-based systems. In de Bakker et al. [dBHRR91], pages 397–446.

[LV93a] N.A. Lynch and F.W. Vaandrager. Forward and backward simulations – part I: Untimed systems. Technical Report MIT/LCS/TM-486, MIT Laboratory for Computer Science, May 1993.

[LV93b] N.A. Lynch and F.W. Vaandrager. Forward and backward simulations – part II: Timing-based systems. Technical Report MIT/LCS/TM-487, MIT Laboratory for Computer Science, September 1993.

[LV95] N.A. Lynch and F.W. Vaandrager. Forward and backward simulations – part II: Timing-based systems. Technical Report CS-R95??, CWI, Amsterdam, ?? 1995.

[Lyn95] N.A. Lynch. *Distributed Algorithms.* Morgan Kaufmann Publishers, Inc., 1995. To appear.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice-Hall International, Englewood Cliffs, 1989.

[Mil93] R. Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25:267–310, 1993.

[MMT91] M. Merritt, F. Modugno, and M. Tuttle. Time constrained automata. In Baeten and Groote [BG91], pages 408–423.

[OL82] S. Owicki and L. Lamport. Proving liveness properties of concurrent programs. *ACM Transactions on Programming Languages and Systems*, 4:455–495, 1982.

[Par81] D.M.R. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, 5^{th} *GI Conference*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer-Verlag, 1981.

[Plo81] G.D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Computer science Department, Aarhus University, 1981.

[Pnu82] A. Pnueli. The temporal semantics of concurrent programs. *Theoretical Computer Science*, 13:45–60, 1982.

[Pnu83] A. Pnueli. On the extremely fair treatment of probabilistic algorithms. In *Proceedings of the* 15^{th} *Annual ACM Symposium on Theory of Computing, Boston, Massachusetts*, May 1983.

[PS95] A. Pogosyants and R. Segala. Formal verification of timed properties of randomized distributed algorithms. In *Proceedings of the* 14^{th} *Annual ACM Symposium on Principles of Distributed Computing,* Ottawa, Ontario, Canada, August 1995. To appear.

[PZ86] A. Pnueli and L. Zuck. Verification of multiprocess probabilistic protocols. *Distributed Computing*, 1(1):53–72, 1986.

[Rab63] M.O. Rabin. Probabilistic automata. *Information and Control*, 6:230–245, 1963.

[Rab76] M.O. Rabin. Probabilistic algorithms. In J. F. Traub, editor, *Algorithms and Complexity: New Directions and Results*, pages 21–39. Academic Press, 1976.

[Rab82] M.O. Rabin. N-process mutual exclusion with bounded waiting by $4 \log N$ shared variables. *Journal of Computer and System Sciences*, 25:66–75, 1982.

[Rao90] J.R. Rao. Reasoning about probabilistic algorithms. In *Proceedings of the* 9^{th} *Annual ACM Symposium on Principles of Distributed Computing,* Quebec, Canada, August 1990.

[Rud66] W. Rudin. *Real Complex Analysis.* McGraw-Hill, 1966.

[Sai92] I. Saias. Proving probabilistic correctness: the case of Rabin's algorithm for mutual exclusion. In *Proceedings of the* 11^{th} *Annual ACM Symposium on Principles of Distributed Computing,* Quebec, Canada, August 1992.

[Sei92] K. Seidel. Probabilistic communicating processes. Technical Report PRG-102, Ph.D. Thesis, Programming Research Group, Oxford University Computing Laboratory, 1992.

[SGG$^+$93] J.F. Søgaard-Andersen, S.J. Garland, J.V. Guttag, N.A. Lynch, and A. Pogosyants. Computer-assisted simulation proofs. In C. Courcoubetis, editor, *Proceedings of the fifth international conference on Computer Aided Verification*, volume 697 of *Lecture Notes in Computer Science.* Springer-Verlag, 1993.

[She87] G.S. Shedler. *Regeneration and Networks of Queues.* Springer-Verlag, 1987.

[SL94] R. Segala and N.A. Lynch. Probabilistic simulations for probabilistic processes. In Jonsson and Parrow [JP94], pages 481–496.

[SL95] R. Segala and N.A. Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing*, 1995.

[SLL93] J.F. Søgaard-Andersen, N.A. Lynch, and B.W. Lampson. Correctness of communication protocols. a case study. Technical Report MIT/LCS/TR-589, MIT Laboratory for Computer Science, November 1993.

[SS90] S. Smolka and B. Steffen. Priority as extremal probability. In Baeten and Klop [BK90], pages 456–466.

[Tof90] C. Tofts. A synchronous calculus of relative frequencies. In Baeten and Klop [BK90].

[Var85] M.Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *Proceedings of 26th IEEE Symposium on Foundations of Computer Science*, pages 327–338, Portland, OR, 1985.

[VL92] F.W. Vaandrager and N.A. Lynch. Action transducers and timed automata. In Cleaveland [Cle92], pages 436–455.

[VW86] M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings Symposium on Logic in Computer Science*, pages 332–344. IEEE Computer Society Press, 1986.

[Whi80] W. Whitt. Continuity of generalized semi-markov processes. *Mathematics of Operations Research*, 5, 1980.

[WLL88] J.L. Welch, L. Lamport, and N.A. Lynch. A lattice-structured proof technique applied to a minimum spanning tree algorithm. Technical Report MIT/LCS/TM-361, MIT Laboratory for Computer Science, June 1988.

[WSS94] S.H. Wu, S. Smolka, and E.W. Stark. Composition and behaviors of probabilistic I/O automata. In Jonsson and Parrow [JP94].

[YCDS94] S. Yuen, R. Cleaveland, Z. Dayar, and S. Smolka. Fully abstract characterizations of testing preorders for probabilistic processes. In Jonsson and Parrow [JP94].

[YL92] W. Yi and K.G. Larsen. Testing probabilistic and nondeterministic processes. In *Protocol Specification, Testing and Verification XII*, pages 47–61, 1992.

[Zuc86] L. Zuck. *Past Temporal Logic.* PhD thesis, The Weizman Institute of Science, 1986.

Table of Symbols

We list the symbols that are used in this thesis in the order they appear in the presentation. Each symbol is listed with a short description and a reference to the pages where it is first defined.

Ω	Sample space.	33
$\mathcal{F}$	σ-field.	33
$\sigma(\mathcal{C})$	σ-field generated by a family of sets $\mathcal{C}$.	33
μ	Measure	33
P	Probability measure.	34
E	Event.	34
$\mathit{completion}()$	Completion of a measure.	34
$\otimes$	Product of σ-fields, of measures, and of discrete probability spaces.	35
$\mid$	Conditional of an event and of an event schema.	36
$\mid$	Conditional of a probabilistic execution fragment.	57
$\mathcal{P}$	Probability space.	37
$\mathcal{D}()$	Dirac distribution.	37
$\mathcal{U}()$	Uniform distribution.	37
$\mathit{Probs}(C)$	Set of discrete probability spaces $(\Omega, \mathcal{F}, P)$ with no 0-probability elements such that $\Omega \subseteq C$.	37
A	Automaton.	37
$\mathit{states}()$	States of.	37
$\mathit{start}()$	Start states of.	37
$\mathit{sig}()$	Action signature of.	37
$\mathit{ext}()$	External actions of.	37
$\mathit{int}()$	Internal actions of.	37
$\mathit{acts}()$	Actions of.	37
$\mathit{trans}()$	Transitions of.	37
$\xrightarrow{a}$	Transition with action a.	38
$\Longrightarrow$	Weak transition.	38
α	Execution (fragment).	39
$\mathit{fstate}()$	First state of.	39
$\mathit{lstate}()$	Last state of.	39
$\mathit{frag}()$	Execution fragments of.	39
$\mathit{exec}()$	Executions of.	39
$\frown$	Concatenation of executions.	39
$\frown$	Transition prefixing operator.	52

$\leq$	Prefix of.	39
$\triangleright$	Suffix operator.	39
$\triangleright$	Transition suffixing operator.	52
β	Trace.	40
$traces()$	Traces of.	40
$\sqsubseteq_T$	Trace preorder.	40
$\parallel$	Parallel composition operator.	41
M	Probabilistic automaton.	46
δ	Termination or deadlock symbol.	46
$ctrans()$	Combined transitions of.	48
H	Probabilistic execution (fragment).	49
$prfrag()$	Probabilistic execution fragments of.	49
$prexec()$	Probabilistic executions of.	49
$\alpha\downarrow$	From an execution of a probabilistic execution fragment to an execution fragment of a probabilistic automaton.	51
$\alpha\uparrow q_0$	From an execution fragment of a probabilistic automaton to an execution of a probabilistic execution fragment.	51
tr	Transition.	51
$\mathcal{P}_{tr}$	Probability space in the transition tr, i.e., $tr = (s, \mathcal{P}_{tr})$ or, if tr is simple, $tr = (s, a, \mathcal{P}_{tr})$.	51
V	Set of actions.	51
U	Set of states.	51
tr_s^M	Transition leaving from state s in the fully probabilistic automaton M.	51
$\mathcal{P}_H$	Probability space associated with the probabilistic execution fragment H.	52
C_α	Cone with prefix α.	53
$\xrightarrow{a}_{\mathrm{C}}$	Combined transition.	58
$\overset{a}{\Longrightarrow}_{\mathrm{C}}$	Weak combined transition.	59
$\mathcal{O}$	Generator of a weak transition.	60
$\upharpoonright$	Action restriction operator.	64
$\lceil$	Projection operator.	65
$\rceil$	Reverse of projection.	66
$Rename_\rho()$	Renaming operator.	72
$Hide_I()$	Hiding operator.	73
$Advs()$	Adversaries for.	80
$prexec(M, \mathcal{A}, \alpha)$	Probabilistic execution fragment of M generated by adversary $\mathcal{A}$ with starting condition α.	80
e	Event schema.	82
$Cones()$	Function that identifies the points of satisfaction of a finitely satisfiable event schema.	83
$\circ_{Cones}$	Concatenation of two event schemas.	83
$\Pr_{Advs,\Theta}(e)\ \mathcal{R}\ p$	Probabilistic statement.	84
$(\equiv, F)$	Oblivious relation.	92
$FIRST(\ldots)$	Coin event: first occurrence of an action among many.	107

$OCC(i,\ldots)$	Coin event: i-th occurrence of an action among many.	109
$GFIRST(\mathcal{S},E)()$	Coin event: first occurrence of an action among many with several outcomes.	122
$GCOIN(\mathcal{S},E)()$	General coin event.	125
$\mathcal{D}$	Trace distribution.	138
$tdistr()$	Trace distribution of.	138
$tdistrs()$	Trace distributions of.	138
$itrace()$	Internal trace of.	139
$itdistr()$	Internal trace distribution of.	139
$itdistrs()$	Internal trace distributions of.	139
$\sqsubseteq_D$	Trace distribution preorder.	141
$\sqsubseteq_{DC}$	Trace distribution precongruence.	143
C_P	Principal context, timed principal context.	145
$ptdistrs()$	Principal trace distributions of.	146
$\sqsubseteq_{\mathcal{R}}$	Lifting of a relation to probability spaces.	168
$\simeq$	Existence of a strong bisimulation.	169
$\sqsubseteq_{\mathrm{SS}}$	Existence of a strong simulation.	169
$\simeq_{\mathrm{P}}$	Existence of a strong probabilistic bisimulation.	171
$\sqsubseteq_{\mathrm{SPS}}$	Existence of a strong probabilistic simulation.	171
$=_{\mathrm{P}}$	Existence of a weak probabilistic bisimulation.	172
$\sqsubseteq_{\mathrm{WPS}}$	Existence of a weak probabilistic simulation.	172
$\sqsubseteq_{FS}$	Existence of a probabilistic forward simulation.	174
$vis()$	Visible actions of.	196
ω	Trajectory.	197
$ltime()$	Last time of.	197
$t\text{-}frag()$	Timed execution fragments of.	199
$t\text{-}exec()$	Timed executions of.	199
$t\text{-}exec_\delta()$	Extended timed executions of.	199
$te\text{-}frag()$	Time-enriched execution fragments of.	201
$te\text{-}prfrag()$	Probabilistic time-enriched execution fragments of.	202
$te\text{-}prexec()$	Probabilistic time-enriched executions of.	202
$sample()$	Function that applied to a probabilistic time-enriched execution H of a probabilistic timed automaton M returns a probabilistic execution H' of M that samples H.	209
$t\text{-}sample()$	Function that applied to a probabilistic time-enriched execution fragment H of a probabilistic timed automaton M returns a probabilistic timed execution fragment H' of M that t-samples H.	211
$\overset{a}{\rightsquigarrow}$	Move.	217
$E_{U,Advs}[e]$	Worst expected time for success of the event schema e starting from a state of U under the action of an adversary from $Advs$.	227
$seq()$	Sequence of a timed sequence pair.	243
$tsp()$	Timed sequence pairs over some given set.	243
$t\text{-}trace()$	Timed trace of.	244
$t\text{-}tdistr()$	Timed trace distribution of.	246
$t\text{-}tdistrs()$	Timed trace distributions of.	247

$\sqsubseteq_{Dt}$	Timed trace distribution preorder.	249
$\sqsubseteq_{DCt}$	Timed trace distribution precongruence.	249
$\mathit{pt\text{-}tdistrs}()$	Principal timed trace distributions of.	250
$\simeq_{\mathrm{Pt}}$	Existence of a probabilistic timed bisimulation.	257
$\sqsubseteq_{\mathrm{Pt}}$	Existence of a probabilistic timed simulation.	258
$\sqsubseteq_{FSt}$	Existence of a probabilistic timed forward simulation.	258

Index