LABORATORY FOR
COMPUTER SCIENCE

MASSACHUSETTS
INSTITUTE OF
TECHNOLOGY

MIT/LCS/TR-477

AD-A223 100

KNOWLEDGE AND DISTRIBUTED COMPUTATION

Mark Tuttle

May 1990

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY 2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
4. PERFORMING ORGANIZATION REPORT NUMBER(S) MIT/LCS/TR-477	5. MONITORING ORGANIZATION REPORT NUMBER(S) N00014-83-K-0125
6a. NAME OF PERFORMING ORGANIZATION MIT Lab for Computer Science 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Office of Naval Research/Dept. of Navy
6c. ADDRESS (City, State, and ZIP Code) 545 Technology Square Cambridge, MA 02139	7b. ADDRESS (City, State, and ZIP Code) Information Systems Program Arlington, VA 22217
8a. NAME OF FUNDING/SPONSORING ORGANIZATION DARPA/DOD 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code) 1400 Wilson Blvd. Arlington, VA 22217	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.
11. TITLE (Include Security Classification) Knowledge and Distributed Computation	
12. PERSONAL AUTHOR(S) Mark Tuttle	
13a. TYPE OF REPORT Technical 13b. TIME COVERED FROM _____ TO _____	14. DATE OF REPORT (Year, Month, Day) May, 1990 15. PAGE COUNT
16. SUPPLEMENTARY NOTATION	
17. COSATI CODES FIELD / GROUP / SUB-GROUP	18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) Knowledge, Distributed Systems, Byzantine Agreement, Distributed Firing Squad, Crytography, Interactive, and Zero Knowledge Proofs.

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Understanding systems of agents that interact in some way is fundamental to many areas of science, including philosophy, linguistics, economics, game theory, logic, artificial intelligence, robotics, and distributed computing. As we try to understand these systems, we often find ourselves reasoning (at least informally) about the *knowledge* these agents have about other agents. Recent work has shown that these informal notions of knowledge can be made precise in the context of computer science. In this thesis, we provide convincing evidence that reasoning in terms of knowledge can lead to general, unifying results about distributed computation, and we extend the standard definitions of knowledge and apply them in new contexts such as cryptography.

Keywords: Cryptography. (KR)

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Judy Little	22b. TELEPHONE (Include Area Code) (617) 253-5894 22c. OFFICE SYMBOL

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

U.S. Government Printing Office: 1985—607-047

90 06 22 196

Many problems in the literature such as the *consensus* and *distributed firing squad* problems require processors in a synchronous system to perform some action simultaneously, yet each problem is solved in each model of processor failure using a different algorithm. We give a single algorithm scheme with which we can transform specifications of such problems directly into protocols that are optimal in a very strong sense: these protocols are *optimal in all runs*, which means that given any possible input to the system and any possible faulty processor behavior, these protocols are guaranteed to perform the simultaneous action as soon as any other protocol would do so in the same context. In contrast, most other protocols in the literature are optimal only in the worst case run. This transformation is performed in two steps. In the first step, we extract directly from the problem specification a high-level protocol programmed using explicit tests for common knowledge. In the second step, we carefully analyze when facts become common knowledge, thereby providing a method of efficiently implementing these protocols in the crash failure model and several variants of the omissions failure model. In the generalized omissions model, however, our analysis shows that testing for common knowledge is NP-hard. Given the close correspondence between common knowledge and simultaneous actions, we are able to show that no optimal protocol for any such problem can be computationally efficient in this model. Our analysis exposes many subtle differences between the failure models, including the precise point at which this gap in complexity occurs. This work shows how knowledge can be effectively used in protocol design and in proving nontrivial lower bounds on computational complexity.

In areas like cryptography, probability often plays a role in understanding interesting systems of agents, yet the standard definition of knowledge used above ignores issues of probability. Recent papers have shown that more than one definition of *probabilistic knowledge* is reasonable, but they do not tell us how to make the choice between these definitions. We clarify the issues involved in making the right choice. We show that no single definition is appropriate in all contexts. Given a particular context, however, we show how to construct the most appropriate definition for that context, where "most appropriate" is made precise in terms of betting games against an adversary. We show how probabilistic knowledge can be used to specify *coordinated attack*, and how different definitions of probabilistic knowledge result in different levels of guarantees by the problem statement. Another important aspect of cryptography is the fact that an agent's knowledge (of the contents of a message, for example) is limited by the bounds on its computational power, yet the standard definition of knowledge ignores computational complexity, in addition to probability. We show how such issues in cryptography motivate the definition of *practical knowledge*, and then turn to the problem of using probabilistic and practical knowledge to reason about cryptography.

While the intuition underlying a zero knowledge proof system [GMR89] is that no "knowledge" is leaked by the prover to the verifier, researchers are just beginning to analyze such cryptographic systems in terms of formal notions of knowledge. We show how the definition of an *interactive proof system* can be characterized directly in terms of practical knowledge. Using this notion of knowledge, we formally capture and prove the intuition that the prover does not leak any knowledge of any fact (other than the fact being proven) during a *zero knowledge proof*. We extend this result to show that the prover does not leak any knowledge of how to compute any information (such as the factorization of a number) during a zero knowledge proof. Finally, we show how our knowledge-theoretic characterization of interactive proof systems can be used to prove simple properties of such systems. This work represents a first step toward the ultimate goal of being able to reason about cryptographic systems directly in terms of knowledge, reasoning at a higher semantic level than the operational cryptographic definitions themselves.

Knowledge and Distributed Computation

by

Mark R. Tuttle

B.S., University of Nebraska–Lincoln
(1984)
M.S., Massachusetts Institute of Technology
(1987)

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

Doctor of Philosophy

at the

Massachusetts Institute of Technology

September 1989

©Massachusetts Institute of Technology, 1989.

Signature of Author _____
Department of Electrical Engineering and Computer Science
September 8, 1989

Certified by _____
Nancy A. Lynch
Professor
Thesis Supervisor

Accepted by _____
Arthur C. Smith
Chairman, Departmental Committee on Graduate Students

DTIC COPY INSPECTED 6

Accesion For
NTIS CRA&I
DTIC TAB
Unannounced
Justification
By
Distribution /
Availability Code
Dist | Avail and / or Special
A-1

Knowledge and Distributed Computation

by

Mark R. Tuttle

Submitted to the
Department of Electrical Engineering and Computer Science
on September 8, 1989,
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

Abstract

Understanding systems of agents that interact in some way is fundamental to many areas of science, including philosophy, linguistics, economics, game theory, logic, artificial intelligence, robotics, and distributed computing. As we try to understand these systems, we often find ourselves reasoning (at least informally) about the *knowledge* these agents have about other agents. Recent work has shown that these informal notions of knowledge can be made precise in the context of computer science. In this thesis, we provide convincing evidence that reasoning in terms of knowledge can lead to general, unifying results about distributed computation, and we extend the standard definitions of knowledge and apply them in new contexts such as cryptography.

Many problems in the literature such as the *consensus* and *distributed firing squad* problems require processors in a synchronous system to perform some action simultaneously, yet each problem is solved in each model of processor failure using a different algorithm. We give a single algorithm scheme with which we can transform specifications of such problems directly into protocols that are optimal in a very strong sense: these protocols are *optimal in all runs*, which means that given any possible input to the system and any possible faulty processor behavior, these protocols are guaranteed to perform the simultaneous action as soon as any other protocol would do so in the same context. In contrast, most other protocols in the literature are optimal only in the worst case run. This transformation is performed in two steps. In the first step, we extract directly from the problem specification a

high-level protocol programmed using explicit tests for common knowledge. In the second step, we carefully analyze when facts become common knowledge, thereby providing a method of efficiently implementing these protocols in the crash failure model and several variants of the omissions failure model. In the generalized omissions model, however, our analysis shows that testing for common knowledge is NP-hard. Given the close correspondence between common knowledge and simultaneous actions, we are able to show that no optimal protocol for any such problem can be computationally efficient in this model. Our analysis exposes many subtle differences between the failure models, including the precise point at which this gap in complexity occurs. This work shows how knowledge can be effectively used in protocol design and in proving nontrivial lower bounds on computational complexity.

In areas like cryptography, probability often plays a role in understanding interesting systems of agents, yet the standard definition of knowledge used above ignores issues of probability. Recent papers have shown that more than one definition of *probabilistic knowledge* is reasonable, but they do not tell us how to make the choice between these definitions. We clarify the issues involved in making the right choice. We show that no single definition is appropriate in all contexts. Given a particular context, however, we show how to construct the most appropriate definition for that context, where "most appropriate" is made precise in terms of betting games against an adversary. We show how probabilistic knowledge can be used to specify *coordinated attack*, and how different definitions of probabilistic knowledge result in different levels of guarantees by the problem statement. Another important aspect of cryptography is the fact that an agent's knowledge (of the contents of a message, for example) is limited by the bounds on its computational power, yet the standard definition of knowledge ignores computational complexity, in addition to probability. We show how such issues in cryptography motivate the definition of *practical knowledge*, and then turn to the problem of using probabilistic and practical knowledge to reason about cryptography.

While the intuition underlying a zero knowledge proof system [GMR89] is that no "knowledge" is leaked by the prover to the verifier, researchers are just beginning to analyze such cryptographic systems in terms of formal notions of knowledge. We show how the definition of an *interactive proof system* can be characterized directly in terms of practical knowledge. Using this notion of knowledge, we formally capture and prove the intuition that the prover does not leak any knowledge of any fact (other than the fact being

proven) during a *zero knowledge proof.* We extend this result to show that the prover does not leak any knowledge of how to compute any information (such as the factorization of a number) during a zero knowledge proof. Finally, we show how our knowledge-theoretic characterization of interactive proof systems can be used to prove simple properties of such systems. This work represents a first step toward the ultimate goal of being able to reason about cryptographic systems directly in terms of knowledge, reasoning at a higher semantic level than the operational cryptographic definitions themselves.

Keywords: knowledge, distributed systems, Byzantine agreement, distributed firing squad, crytography, interactive and zero knowledge proofs

Thesis supervisor: Nancy A. Lynch
Title: Professor

Contents

Acknowledgements **7**

1 Introduction **11**
1.1 Motivation 12
1.2 Related Work 14
1.3 Thesis Contributions 17

2 Knowledge and Common Knowledge **21**
2.1 Systems of Agents 21
2.2 Definition of Knowledge 23
2.3 Logic of Knowledge 29
2.4 Properties of Knowledge 30

3 Programming Simultaneous Actions using Common Knowledge **35**
3.1 Introduction 35
3.2 Model of a System 40
3.3 Simultaneous Choice Problems 45
3.4 Optimal Protocols 49
3.5 Testing for Common Knowledge 56
3.5.1 The Omissions Model 61
3.5.2 Receiving Omissions 80
3.5.3 Generalized Omissions 82
3.6 Conclusions 95

4 Knowledge, Probability, and Adversaries **99**
4.1 Introduction 99

4.2 Probability on runs 103
4.3 Probability at a point 109
4.4 Definitions of probabilistic knowledge 111
4.5 Probability in synchronous systems 117
4.6 Probability in asynchronous systems 128
4.7 An application: coordinated attack 135
4.8 Conclusion 139
4.A Proofs of results 139
4.B Discussion 149
4.B.1 The need for protocols 149
4.B.2 Safe bets and nonmeasurable facts 151

5 Knowledge and Zero Knowledge 153
5.1 Introduction 153
5.2 Interactive and Zero Knowledge Proof Systems 159
5.2.1 Interactive protocols 159
5.2.2 Interactive proof systems 161
5.2.3 Zero knowledge proof systems 165
5.3 Knowledge 169
5.3.1 Knowledge and Probability 170
5.3.2 Knowledge and Computation 171
5.4 Knowledge and Interactive Proofs 182
5.5 Knowledge and Zero Knowledge 186
5.6 Generation and Zero Knowledge 189
5.7 Resource-bounded provers 193
5.8 An Application 201
5.9 Conclusion 204
5.A Proofs of results 206

6 Conclusion 221

Bibliography 225

Index 235

List of Figures

3.1 Communication graphs. 54
3.2 Runs illustrating Lemma 3.9. 65
3.3 Runs illustrating Lemma 3.10. 67
3.4 An example of the construction when $t = 9$. 69
3.5 Embedding a graph G in a run r. 90

4.1 A (labeled) computation tree. 107

Acknowledgements

I am grateful to have had Nancy Lynch as my thesis advisor. Exciting things happen when Nancy is around. Her enthusiasm, tempered with a healthy dose of skepticism, produces a very stimulating research environment, and attracts an inspiring collection of postdoctoral and graduate students to MIT to work with her. I am grateful to have been a part of this.

I am also grateful to the other members of my thesis committee, Shafi Goldwasser, Joe Halpern, and Yoram Moses, for the stimulation and encouragement they have provided. I am particularly grateful to Joe and Yoram, both collaborators on work in this thesis. During his stay as a visitor at MIT, Yoram became a close friend and collaborator, and had a profound influence on how I do research, what problems I choose to work on, and how I express my solutions. Later, as a visitor myself at the IBM Almaden Research Center, a similar relationship began to develop with Joe. Their contrasting approaches to research and expression have been very stimulating. I am grateful to them both.

I thank a number of people who have contributed in significant ways to the work in this thesis, either directly through me or indirectly through Joe and Yoram. These include Hagit Attiya, Paul Beame, Brian Coan, Cynthia Dwork, Ron Fagin, Alan Fekete, Oded Goldreich, Adam Grove, Vassos Hadzilacos, Amos Israeli, Michael Merritt, Albert Meyer, David Peleg, Larry Stockmeyer, Moshe Vardi, and Jennifer Welch.

I thank my parents, Morrie and Amy, and my sister, Caroline, for their love and support. Above all, I thank my wife, Margaret, who has been a constant source of happiness and inspiration.

I have been supported during this work by a GTE Graduate Fellowship and by an IBM Graduate Fellowship. I would also like to thank the IBM Almaden Research Center for their support during two summer visits, and

the DEC Cambridge Research Lab for their support during the final weeks of preparing this thesis. I have also been supported in part by the National Science Foundation under Grant CCR-86-11442, by the Office of Naval Research under Contract N00014-85-K-0168, and by the Defense Advanced Research Projects Agency (DARPA) under Contract N00014-83-K-0125.

To Margaret

Chapter 1

Introduction

Today, with the exception of home personal computers, nearly every computer is part of a larger network of computers. A *distributed system* is a collection of computers (or processors) that can exchange information by sending message to one another over some communication network. The motivation behind building a distributed system may be as simple as the desire to allow people working at the computers to send messages to each other, or to share the use of a common printer. A more sophisticated reason for doing so is to allow the computers to work together to solve a problem. See #1473

Unfortunately, writing the program to solve this problem is often quite difficult. This is usually because the problem is defined in terms of the global system state, whereas an individual processor must base its actions solely on the information recorded in its local state, typically a small fraction of the information represented by the global state. As a result, a processor must base its actions on incomplete knowledge of the global state. The limitations of what a processor can know about the global state is a fundamental source of difficulty when programming distributed systems. It often feels quite natural, therefore, to reason informally about distributed computation in terms of what each processor knows. The primary purpose of this work is to explore the role of *knowledge* in the design and analysis of *distributed algorithms*. We provide some convincing evidence that reasoning in terms of knowledge can yield general, unifying results about distributed computation, and we extend the standard definitions of knowledge in order to apply them in new contexts.

1.1 Motivation

One of the most well-known examples of informal reasoning about knowledge when thinking about distributed computing involves the *coordinated attack* problem, a formulation by Gray [Gra78] of a folk theorem concerning the impossibility of coordination in asynchronous systems. This problem is defined as follows. Two generals A and B are on opposite hills with a common enemy encamped in the valley between them. Neither general has any initial intention of attacking, but might at some later point decide to attack the enemy. The two generals must attack the enemy simultaneously, however, since a general attacking by himself is certain to be destroyed. Unfortunately, the only way the two generals can communicate is via messengers who may be captured enroute by the enemy. The coordinated attack problem is the following: is there a protocol the two generals can follow that guarantees both generals attack the enemy simultaneously whenever a single general attacks?

Gray shows that the only such protocol is one in which neither general attacks. To see this, suppose P is a protocol for coordinated attack, and suppose there is an execution of P in which the two generals attack simultaneously after exchanging a total of k messages (that is, after dispatching k messengers who may or may not have successfully delivered their messages). Consider the last message m received by either of the generals before the attack. Suppose m was sent by general A, and consider the instant the attack begins. At this point, A doesn't know whether B has received m or not, but A has committed himself to the attack in either case. If we consider the execution differing from the current execution only in that B does *not* receive m, therefore, we see that A also attacks. Since P guarantees that both generals attack whenever a single general attacks, this must be an execution of this protocol in which the two generals attack simultaneously after exchanging only $k-1$ messages. Continuing by induction, we see that if there is any execution of P in which the two generals attack, then there is an execution in which the two generals attack without sending any messages. But if no messages are sent, then B cannot possibly know of A's intention of attacking, and a simultaneous attack is impossible. It follows that the only protocols for coordinated attack are protocols in which neither general attacks!

This appeal to our intuition that A does not "know" whether B has received m seems quite natural. Roughly speaking, from A's point of view, there are two global states consistent with the information recorded in A's

local state: either B has received m, or the messenger carrying m was captured by the enemy and B has not received m. It follows that A cannot *know* that m has been received, since it is *possible* that B has not received m. Philosophers have formalized this intuition concerning knowledge as early as 1962 with Hintikka's *possible world semantics* for knowledge [Hin62]. The basic idea is that, in any world or state of affairs, a processor considers a number of worlds to be possible in addition to the actual world, and that a processor *knows* a fact if that fact is true in all worlds the processor considers possible. In the case of coordinated attack, for example, A considers at least two worlds possible, one in which m was received and one in which m was not, and hence cannot be said to know m has been received since one of the worlds it considers possible is a world in which m has not been received.

An interesting difference between the use of knowledge by philosophers and by computer scientists, however, is that computer scientists tend to be interested in the knowledge of groups of processors as well as the knowledge of individual processors. For example, we can say that *everyone knows* a fact if every processor knows the fact according to the definition of knowledge given above. Another interesting state of knowledge turns out to be the state of *common knowledge*. Roughly speaking, a fact is common knowledge if everyone knows the fact, everyone knows that everyone knows the fact, and so on. Such definitions of knowledge were first made in the context of distributed computing by Halpern and Moses [HM84] (and later by others [CM86, FI86, PR85]). In fact, in that paper they give a formal proof of the impossibility of coordinated attack directly in terms of knowledge. They show that attaining common knowledge of a certain fact is a *necessary* condition for the generals to attack. They go on to prove, using an argument very similar to the combinatorial argument sketched above, that it is impossible to attain common knowledge of any nontrivial fact in (asynchronous) systems where messages (or messengers) can be lost or indefinitely delayed. Combining these results, it follows that coordinated attack is impossible in such systems.

This argument is a rigorous proof that captures much of the informal intuition concerning knowledge in the proof sketched above. In distributed computing, when an algorithm or an impossibility proof is sketched, it is often the appeal to our intuition concerning knowledge that makes the presentation understandable. When this sketch is made rigorous, however, it typically does not make explicit references to any notion of knowledge, and this intuition that was so helpful before is now buried under complex, combi-

natorial arguments. Halpern and Moses made a fundamental contribution in showing that it is possible to make rigorous the intuition concerning knowledge we use informally when reasoning about distribute algorithms. As a result, they made significant progress toward the goal of making explicit reasoning about knowledge a fundamental tool for reasoning about distributed computation. Part of the motivation for this work is to make further progress toward this goal.

1.2 Related Work

By far the most common use of knowledge in distributed computation has been to prove lower bounds and impossibility results. A fundamental technique for proving lower bounds on message complexity is given by Chandy and Misra [CM86], where they analyze the communication complexity required for a processor to reach a given state of knowledge in an asynchronous system. Roughly speaking, they show that if at time t processor i_1 does *not* know a fact φ, and at a later time t' processor i_m knows processor i_{m-1} knows ... processor i_1 *does* know φ, then some sequence (or chain) of messages from i_1 to i_2 ... to i_m must have occurred between times t and t'. Using this result, they show how to prove lower bounds on communication complexity for various problems such as *mutual exclusion* and *termination detection*. These proofs proceed by showing that a certain number of levels of "processor i knows processor j knows" are required to solve the problem, and then appealing to their main theorem to prove that any protocol solving the problem must result in a chain of messages of a certain length.

Along the same lines, Moses and Roth have recently performed a slightly more sophisticated analysis in [MR89] where they study the problem of message diffusion in asynchronous systems [SFC85], the problem of diffusing a given message throughout a system in such a way that each processor "consumes" the message exactly once. They show that two levels of knowledge are sufficient if communication in the system is not required to subside, and that any subsiding protocol must either attain three levels of knowledge or use three different types of messages. Lower bounds on message complexity of such protocols follow immediately.

Similarly, in [Had87], Hadzilacos studies two- and three-phase atomic commit protocols (used in the context of transaction processing in distributed

databases) in terms of knowledge, and characterizes the levels of knowledge required for a site to commit a transaction when following such protocols. As corollaries of these characterizations, he is able to show that no nonblocking atomic commit protocol can tolerate communication failures, and he is able to derive a known lower bound (due to Dwork and Skeen [DS83]) on the number of messages required to commit a transaction. In the same vein, Mazer [Maz88, Maz89] performs a knowledge-theoretic analysis of commit protocols that guarantee that all participants reach a consistent decision on the commitment of a transaction in systems where failed sites can recover and rejoin the system.

As with coordinated attack, a number of impossibility results for computation in asynchronous systems follow from the fact that common knowledge cannot be attained in such systems. But some problems *can* be solved in asynchronous systems. This implies that the state of common knowledge is not relevant in the context of these problems. In order to analyze these problems, therefore, a number of other definitions of knowledge such as *eventual common knowledge* and *time-stamped common knowledge* have been proposed (see [HM84]). In [PT88], Panangaden and Taylor define the notion of *concurrent common knowledge* and show how several problems such as finding global snapshots [CL85] of the global system state can be analyzed in terms of concurrent common knowledge.

Just as important as lower bounds and impossibility results, however, is the use of knowledge in the actual design of protocols. The motivation for the use of knowledge in protocol design is that a processor's actions must depend on what it knows. When a protocol tests for the equality of two variables, the protocol is implicitly testing for a certain state of knowledge. In [HF88], Halpern and Fagin generalize the standard notion of a protocol by defining *knowledge-based* protocols, protocols in which a processor's actions may explicitly depend on tests for knowledge. Such protocols typically include explicit tests for knowledge, and include statements such as "if processor 1 knows processor 2 has received message m, then perform action a." Translating a knowledge-based protocol into a standard protocol, therefore, requires implementing the embedded tests for conditions such as "processor 1 knows processor 2 has received m." The advantage of knowledge-based protocols, however, is that they often provide a simple, high-level description and explanation of a processor's behavior. For example, Halpern and Zuck construct in [HZ87] a family of knowledge-based

protocols solving the sequence transmission problem (the problem of transmitting a sequence of bits over an unreliable channel), and show that known solutions [AUY79, AUWY82, BSW69, Ste76] to the sequence transmission problem, including the alternating bit protocol, can be viewed as particular instances of these knowledge-based protocols.

Another example of the useful level of abstraction knowledge-based protocols provide is the work of Neiger and Toueg in [NT87]. They construct a broadcast primitive that can be used to cause certain facts to become "common knowledge" in systems with asynchronous communication, systems in which true common knowledge cannot be attained. Consequently, using this tool (and other tools developed in the paper), programmers are able to make simplifying assumptions when they design protocols by assuming common knowledge of certain facts is attainable, and are able to implement these protocols using these broadcast primitives.

The first significant use of knowledge in the design of new protocols, however, is the work of Dwork and Moses in [DM86]. They study the problem of *simultaneous Byzantine agreement* [PSL80, Fis83] in which each processor starts with an initial input bit, and all processors are required to come to agreement on a final output bit simultaneously at some later time. They analyze this problem in synchronous systems with the *crash failure model*, a simple failure model in which a processor may crash in the middle of an execution and never again participate in that execution. They show that in such systems common knowledge of a certain fact is a necessary and sufficient condition for processors to reach agreement. Using this observation, they construct a knowledge-based protocol that is optimal in a very strong sense: this protocol is *optimal in all runs*, which means that given any possible input to the system and any possible faulty processor behavior, this protocol is guaranteed to reach consensus soon as any other protocol would do so in the same context. In contrast, most protocols in the literature perform in every run only as well as they do in their worst case run. The protocol constructed in [DM86] for agreement, for example, can halt in as few as two rounds of communication, much sooner that most known protocols. They then construct polynomial-time implementations of the tests for common knowledge embedded in their knowledge-based protocol, resulting in a standard (optimal) protocol for agreement.

1.3 Thesis Contributions

The results of Dwork and Moses are the springboard for the first half of this work. In Chapter 3, we generalize their work in several dimensions.

While Dwork and Moses show how to construct optimal protocols for agreement, implicit in their work is a technique for constructing optimal protocols for many other problems such as the *distributed firing squad* problem, problems in which processors are required to choose and perform the same action simultaneously. In order to make this precise, we define the general class of *simultaneous choice problems*. Problems in this class, including the agreement and distributed firing squad problems, require processors to choose and perform a *simultaneous action*, an action (such as deciding on the value of an output bit) that must be performed simultaneously by all processors whenever it is performed by any processor. In the literature, each combination of a simultaneous choice problem and a failure model results in a different algorithm. In contrast, we give a single algorithm scheme with which we can transform specifications of such problems directly into protocols that are optimal in all runs, in the sense of Dwork and Moses, in a number of failure models. This transformation is performed in two steps. In the first step, we extract directly from the problem specification a high-level protocol programmed using explicit tests for common knowledge. In the second step, we carefully analyze when facts become common knowledge, resulting in efficient implementations of the tests for common knowledge embedded in this high-level protocol, and consequently providing a method for efficiently implementing these protocols.

The high-level, knowledge-based protocols we construct are similar to the protocol given by Dwork and Moses. The technical analysis we perform in order to implement the embedded tests for common knowledge, however, is quite different. The analysis of Dwork and Moses makes strong use of particular properties of the crash failure model and does not extend to more complicated failure models. In contrast, our analysis applies to both the crash failure model and several variants of the *omissions failure model*, a model in which faulty processors may intermittently fail to send messages, instead of crashing at some point and falling silent from then on. Interestingly, our techniques for implementing tests for common knowledge are purely combinatorial. As a result, our work is a nice example of how knowledge-theoretic and combinatorial reasoning can be used together in protocol design: think-

ing in terms of knowledge allows us to isolate the heart of a problem, which can in turn be solved using combinatorial methods.

Given that similar knowledge-based protocols yield optimal protocols for agreement in both the crash and omissions failure models, one might hope that the same protocol would work in even more malicious models like the Byzantine model where faulty processors are allowed to behave in an arbitrary fashion. We are able to show, however, that this is quite unlikely. We consider a variant of the omissions model called the *generalized omissions model* in which faulty processors may intermittently fail both to send *and* to receive messages. In this model, we show that the same knowledge-based protocol is an optimal protocol for performing simultaneous actions, but that implementing tests for common knowledge in this model is suddenly NP-hard! In fact, using the close correspondence between common knowledge and the performance of simultaneous actions, we are able to show that *any* protocol for performing simultaneous actions in this model that is optimal in all runs must require processors to perform NP-hard computations. This means, for example, that there can be no optimal, polynomial-time protocol for agreement, assuming $P \neq NP$. Our analysis exposes many subtle differences between the failure models we consider, including the precise point at which this gap in complexity occurs. This work shows how knowledge can be effectively used in protocol design, as does the work of Dwork and Moses, but it also shows how knowledge can be used to prove nontrivial lower bounds on computational complexity.

One consequence of this work is that it shows for the first time that definitions of knowledge must take computational complexity into account even when analyzing simple problems in relatively simple failure models, and even when issues of computational complexity have not been introduced artificially via cryptographic assumptions. In general, however, there are many situations in which the standard definition of knowledge does not seem appropriate. One of the important contributions of this thesis is to improve our understanding of how to define notions of knowledge for use in these contexts. This is the topic of the second half of this thesis.

One context in which the standard definition of knowledge does not seem particularly appropriate is the context of probabilistic protocols. Such protocols are quite important in computer science since there are a number of problems (such as testing for primality [Rab80]) that we can solve probabilistically but not deterministically, and we would like to be able to reason about

these protocols in terms of knowledge, too. Probabilistic protocols, however, typically guarantee that certain conditions hold only with high probability, and not with certainty. Consequently, while a processor may not *know* a given fact is true, it may be quite confident the fact is true. In [FH88], Fagin and Halpern give a general framework in which it is possible to define an entire family of definitions of knowledge, called *probabilistic knowledge*, that incorporate knowledge and probability. Their idea essentially depends on being able to assign probability spaces to the various processors to use when computing their "confidence" that a given fact is true. They do not tell us, however, which assignment to use.

In Chapter 4, we show how to construct the "best" assignment of probability spaces, and hence the "best" definition of probabilistic knowledge. Surprisingly, however, one of our main observations is that there is no single definition of probabilistic knowledge that is most appropriate in all contexts. More precisely, we show that the various definitions of probabilistic knowledge can best be understood in terms of betting games and betting against different adversaries. We show how different adversaries lead to different definitions of probabilistic knowledge, and given a particular adversary, we show how to construct the "best" definition of probabilistic knowledge for this particular adversary (where "best" is made precise in terms of betting games). In addition, we show how definitions of probabilistic knowledge can be used to analyze probabilistic protocols: we give a specification of a probabilistic version of coordinated attack in terms of probabilistic knowledge, and then show how different definitions of probabilistic knowledge (corresponding to increasingly powerful adversaries) result in problem specifications with increasingly powerful correctness conditions.

Another context in which the standard definition of knowledge does not seem particularly appropriate is when it is important to recognize the bounds on processors' computational resources. The standard definition of knowledge essentially says that a processor knows any fact that follows from the information in its local state, regardless of the complexity of *computing* that fact. In the context of cryptography, for example, the assumption that a polynomial-time processor cannot factor a random integer and hence cannot know its factorization is often crucial to the security of cryptographic protocols. In fact, cryptographic protocols are interesting because they typically combine both the use of probability and the use of complexity-theoretic assumptions, meaning that a definition of knowledge useful in the context

of cryptography will have to incorporate both probability and bounds on processors' computational resources.

Two types of cryptographic protocols that have received an enormous amount of attention recently are *interactive* and *zero knowledge proof systems* [GMR89]. The intuition underlying a zero knowledge proof system is that a "prover" would like to convince a "verifier" that a certain fact is true without leaking any "knowledge" of any other fact to the verifier in the process. Interestingly, while this intuition is closely related to notions of knowledge, the cryptographic definitions of such proof systems do not make any explicit reference to knowledge.

In Chapter 5, we explore definitions of knowledge that incorporate both probability and bounds on processors' computational powers. In particular, we show how interactive proof systems motivate a new notion of *practical knowledge*. We then characterize the definition of an interactive proof system directly in terms of practical knowledge. Using this definition of knowledge, we capture the intuition that the verifier learns essentially nothing as a result of a zero knowledge proof, other than the fact the prover initially sets out to prove. Finally, using these characterizations, we sketch an example of how to prove simple properties of such proof systems directly in terms of knowledge. This work represents a first step toward the ultimate goal of being able to reason about cryptographic systems directly in terms of knowledge, reasoning at a higher semantic level than the operational cryptographic definitions themselves. In addition, this work sheds some light on issues concerning definitions of knowledge (like practical knowledge) that account for processors' limited computational resources.

Chapter 2

Knowledge and Common Knowledge

In this work, we will study systems of agents that interact in some way, typically to solve a problem. While the precise meaning of an agent will depend on the system under consideration (an agent may be a processor in a distributed system or a consumer in an economic model), the meaning should always be clear from context. The purpose of this chapter is to review the standard definitions of what it means for such an agent to "know" something.

2.1 Systems of Agents

We begin with a formal model of a system of agents. Our model is essentially that of [HF88], a simplification of [HM84].

Consider a system of n interacting agents $p_1, \ldots, p_n$ (we will sometimes denote agents with letters like p and q). Loosely speaking, an interaction of these agents is uniquely determined by the sequence of global states through which the system passes in the course of the interaction. Formally, a ***global state*** is an $(n+1)$-tuple $(s_e, s_1, \ldots, s_n)$ of ***local states***, where s_i is the local state of agent p_i (also called p_i's ***view***) and s_e is the state of the environment.

Much of this chapter's presentation comes from joint work with Yoram Moses [MT86, MT88], which was in turn patterned after [HM84, DM86]. Although the notion of an *indexical set* was first defined in [MT86, MT88], the basic ideas used in the proofs in this chapter have appeared elsewhere [HM84, HM85].

Intuitively, the state of the environment is intended to capture everything relevant to the state of the system that cannot be deduced from the agents' local states. In a message-passing system, for example, the state of the environment might include a message buffer for each processor in the system, containing the messages sent to the processor but not yet delivered. A *run* is an infinite sequence of global states; numbering the states from 0 to infinity, we think of the kth global state as the global state at time k. Intuitively, a run is a complete description of one possible interaction of the system agents. A *system* is simply a set of runs, describing the set of all possible interactions of the system agents (possibly the set of all possible executions of a given protocol, for example). We denote the global state at time k in run r by $r(k)$, the local state of p_i in $r(k)$ by $r_i(k)$ (when denoting p_i by q, we denote q's local state by $r_q(k)$), and the state of the environment in $r(k)$ by $r_e(k)$. We refer to the ordered pair (r, k) consisting of a run r and a time k as a *point*. We say that a point (r, k) is a point of a system $\mathcal{R}$ iff r is a run of $\mathcal{R}$, and we frequently abuse notation and write $(r, k) \in \mathcal{R}$ to denote the fact that (r, k) is a point of $\mathcal{R}$. Finally, for notational convenience, we often denote points with letters like c or d.

We typically assume that all agents in a system are following some sort of *protocol* which, roughly speaking, determines an agent's behavior as some function of its local state. This assumption is particularly important in Chapters 3 and 5. Since the systems considered in these chapters are synchronous, we now give a general definition of a protocol in a synchronous system which will be refined later in these chapters.

To motivate the definition of a protocol, consider the following informal description of computation in a synchronous system of agents following a protocol P. Computation begins in an initial state at time 0 and proceeds in a sequence of rounds, with round k lasting from time k through $k+1$ (time k is considered to be part of the preceding round $k-1$). Round k consists of three phases. First, each agent performs some action (such as deciding on an output value) and sends messages to other agents in the system as determined by the protocol P and its local state at time $k-1$. Next, each agent receives all messages sent to it during round k by other agents in the system. Finally, each agent changes its local state as determined by the protocol P, its local state at time $k-1$, and the messages it received during round k.

Formally, therefore, a *protocol* is a tuple of *local protocols*, one for each

agent. A *local protocol* for an agent consists of three components: a function called an *action protocol* that maps a local state to an action a, where a is intuitively the action the agent is to perform in the local state; a function called a *message protocol* that maps a local state to a list $m_1, \ldots, m_n$ of messages, where m_i is intuitively the message to be sent to p_i in the local state; and a function called a *state protocol* that maps a local state and a list $m_1, \ldots, m_n$ of n messages to another local state, where m_i is intuitively the message just received from p_i. A protocol is a *deterministic* protocol if these functions are deterministic, and a *probabilistic* protocol if these functions are probabilistic. We implicitly associate with a protocol a collection of global states called *initial states.*

A run r of a protocol P, sketched informally above, can be captured in terms of our formal definition of a run as follows. The global state of r at time 0 is an initial state. The local state of agent p_i at time $k > 0$ is determined as follows: first, for each agent p_j, apply p_j's message protocol to its local state at time $k-1$ to determine what message p_j sends to p_i during round k, and then apply p_i's state protocol to its local state at time $k-1$ and this set of messages to determine p_i's local state at time k. It is technically convenient to assume that the state of the environment at each time $k \geq 0$ encodes the protocol P and the *history* of the run through time k, where the history is a list of $k+1$ n-tuples giving the local state of each processor at each time from 0 through k. Given a protocol P and a run r as defined above, we say that *r is a run of P*. We note, however, that in later chapters it will be necessary to elaborate this definition of a run of P. For example, in Chapter 3 agents will be able to receive messages from sources outside the system in addition to agents within the system. Furthermore, in that chapter we will consider unreliable systems in which some messages may fail to be delivered, meaning that the global state at time k is not necessarily uniquely determined by the global state at time $k-1$ as defined above.

2.2 Definition of Knowledge

Having defined a system of agents, let us fix a given system $\mathcal{R}$ for the remainder of this section. We are now in a position to say what it means for an agent of $\mathcal{R}$ to know that a given fact is true.

Before we do so, however, we must say what we mean by a fact. Infor-

mally, a fact is an assertion that is either true or false at a point. Formally, we identify a fact φ with a set of points of $\mathcal{R}$, intuitively the set of points at which φ is true, and we write $c \models \varphi$ iff φ is true at c.

The basic intuition behind the definition of knowledge [HM84] is that p_i's local state at c captures all the information p_i has about the system at c. If p_i has the same local state at two points c and d, then at point c agent p_i cannot distinguish between c and d and must consider both as possible candidates for the current point. If a fact φ is true at c but false at d, then p_i cannot be said to *know* at c that φ is true since it is *possible*, from p_i's point of view, that the current point is actually d where φ is false, and not c. This intuition leads us to say that p_i *considers* d *possible at* c if p_i has the same local state at c and d (that is, p_i considers (r', k') possible at (r, k) if $r_i(k) = r'_i(k')$), and that p_i *knows* a fact φ at c if φ is true at all points p_i considers possible at c. In other words, p_i knows φ iff φ is guaranteed to hold, given the information recorded in p_i's local state. We denote "p_i *considers* d *possible at* c" by $c \sim_i d$, and "p_i *knows* φ *at* c" by $c \models K_i\varphi$. It follows that

$$c \models K_i\varphi \text{ iff } d \models \varphi \text{ for all } d \in \mathcal{R} \text{ satisfying } c \sim_i d.$$

Notice that p_i's knowledge depends on the system $\mathcal{R}$, since $\mathcal{R}$ restricts the set of points p_i considers possible. Typically, however, the system will be clear from context. When the system is *not* clear from context, we write $\mathcal{R}, c \models K_i\varphi$ instead of $c \models K_i\varphi$.

Many times we are interested in the knowledge not just of an individual agent, but groups of agents. A straightforward generalization of an individual agent's knowledge is *implicit knowledge* [HM84] (also called *distributed knowledge*). The intuition here is that, just as an individual agent considers many points possible at c, a group of agents pooling together all the information they have about the system may also consider a number of different points possible; and just as the individual agent knows φ if φ holds at all points it considers possible, the group of agents implicitly knows φ if φ holds at all points the group jointly considers possible. Formally, we define the *joint view* of a group G of agents at a point (r, k) by

$$r_G(k) \stackrel{def}{=} \{\langle p_i, r_i(k)\rangle : p_i \in G\}.$$

Roughly speaking, G's view is simply the joint view of its members. We note that it is important to take this joint view to be ordered pairs of the form

$\langle p_i, r_i(k)\rangle$ since we have not said an agent's local state contains its identity, and we want $r_G(k) = r'_G(k')$ to mean every agent in G has the same local state in $r(k)$ and $r'(k')$. We say a group G ***considers a point d possible at*** c if every agent in G considers d possible at c; that is, G considers (r', k') possible at (r, k) iff $r_G(k) = r'_G(k')$. We denote "*G **considers d possible at** c*" by $c \sim_G d$, and "*G **implicitly knows** φ **at** c*" by $c \models I_G\varphi$. We define G's implicit knowledge by

$$c \models I_G\varphi \text{ iff } d \models \varphi \text{ for all } d \in \mathcal{R} \text{ satisfying } c \sim_G d.$$

Intuitively, G implicitly knows φ if the joint view of G's members guarantees that φ holds. If p_i knows ψ and p_j knows $\psi \supset \varphi$, for example, then together they implicitly know φ, even if neither of them knows φ individually.

With these definitions we can make formal sense of statement such as "p_i knows φ," but we can also make sense of statements such as "p_j knows p_i knows φ" involving multiple levels of knowledge. Continuing in this way, we reach in the limit the state of *common knowledge* [HM84]. Roughly speaking, a fact φ is common knowledge to a group of agents if everyone in the group knows φ, everyone knows everyone knows φ, and so on *ad infinitum*. The state of common knowledge will be central to our analysis in Chapter 3. Its central role will result from the close correspondence between common knowledge among the members of a group of processors in a distributed system and the simultaneous performance of an action by members of this group.

The first step in defining common knowledge is to define what it means for everyone in a group to know a fact. For a fixed group G of agents, the standard definition [HM84] of *everyone in G knows φ* is given by

$$E_G\varphi \stackrel{\text{def}}{=} \bigwedge_{p_i \in G} K_i\varphi.$$

The definition [HM84] of *φ is common knowledge to G*, therefore, is given by

$$C_G\varphi \stackrel{\text{def}}{=} E_G\varphi \wedge E_G E_G\varphi \wedge \cdots \wedge E_G^k\varphi \wedge \cdots.$$

Here we define $E_G^k\varphi$ inductively by $E_G^0\varphi = \varphi$ and $E_G^k\varphi = E_G(E_G^{k-1}\varphi)$ for $k \geq 1$. In other words, $c \models C_G\varphi$ iff $c \models E_G^k\varphi$ for all $k \geq 1$. Thus, roughly speaking, a fact is common knowledge if everyone knows it, everyone knows that everyone knows it, and so on *ad infinitum*.

In practice, however, the group of interest will not be a fixed set of agents. For example, in Chapter 3 we will be most interested in facts that are common knowledge to the group $\mathcal{N}$ of nonfaulty processors. The precise meaning of a nonfaulty processor is not important here, so we do not define $\mathcal{N}$ formally at this point; simply observe that a processor may be considered faulty at some points and not at others, and hence that the set of "nonfaulty processors" is not a constant, fixed set of processors but varies from point to point. This motivates the definition of common knowledge to a slightly more general notion of groups of agents. An *indexical set* $\mathcal{S}$ of agents is a function mapping points to sets of agents (meaning $\mathcal{S}$ is a set whose value is indexed by points, so to speak). That is, $\mathcal{S} : c \mapsto \mathcal{S}(c)$, where $\mathcal{S}(c)$ is a set of agents. The notion of an indexical set is a direct generalization of the notion of a fixed set of agents. In particular, we can identify a fixed set of agents with a constant indexical set. The group $\mathcal{N}$ of nonfaulty processors, the group P of all processors, the group of all processors that haven't displayed faulty behavior by the current time, and many other groups of interest are all indexical sets of processors. In practice, each of these indexical sets is nonempty. For example, since it is common in the literature to assume that the upper bound on the number of faulty processors to be tolerated is strictly less that the number of processors in the system, the set of nonfaulty processors is always nonempty. Formally, an indexical set $\mathcal{S}$ is *nonempty* (in a given system $\mathcal{R}$) if $\mathcal{S}(c)$ is nonempty for every point c of $\mathcal{R}$. For technical convenience, we restrict our attention to nonempty indexical sets.

The first step in defining what it means for a fact φ to be common knowledge to agents in an indexical set is to define what it means for everyone in the indexical set to know φ. In extending the standard definition to indexical sets, a subtle decision must be made. The immediate generalization is to define

$$E_{\mathcal{S}}\varphi \stackrel{\text{def}}{=} \bigwedge_{p_i \in \mathcal{S}} K_i\varphi.$$

This means that $c \models E_{\mathcal{S}}\varphi$ iff $c \models K_i\varphi$ for every $p_i \in \mathcal{S}(c)$. This generalization, however, does not capture a subtle aspect of agents' knowledge in unreliable systems. Consider, for example, a system with some action a in which it is guaranteed that all nonfaulty processors perform a simultaneously whenever any nonfaulty processor does so. (Again, the precise definition of a nonfaulty processor is not important here.) Suppose the nonfaulty processors perform a at a point c. It seems reasonable to expect that at the point

c all nonfaulty processors know that all nonfaulty processors are performing a; in other words, $c \models E_{\mathcal{N}}\varphi$ where φ is the fact "*all nonfaulty processors are performing a.*" The reasoning is as follows: each nonfaulty processor is performing a, so each nonfaulty processor knows a is being performed by a nonfaulty processor; and since a is guaranteed to be performed simultaneously by all nonfaulty processors whenever it is performed by any nonfaulty processor, each nonfaulty processor knows all nonfaulty processors are performing a. This line of reasoning, however, depends on a nonfaulty processor knowing it *is* a nonfaulty processor, which need not be the case (and it certainly won't be the case in Chapter 3). The only thing a nonfaulty processor really knows at the point c is that *if* it is nonfaulty, *then* the action a is being performed by all nonfaulty processors.

While it is possible for a nonfaulty processor to be a member of the indexical set $\mathcal{N}$ without knowing it is a member of $\mathcal{N}$, it is not hard to see that for any *fixed* (or constant) set G, an agent is a member of G iff it knows it is a member of G. This follows directly from the definition of knowledge, since if $p_i \in G$, then $p_i \in G$ holds at all points (and in particular at all points p_i considers possible), and hence p_i knows $p_i \in G$. Similarly, given an agent $p_i \in G$, it is not hard to see that p_i knows φ iff p_i knows $(p_i \in G) \supset \varphi$: if p_i knows φ, then φ, and hence $(p_i \in G) \supset \varphi$, holds at all points p_i considers possible, and therefore p_i knows $(p_i \in G) \supset \varphi$. An equivalent definition of $E_G\varphi$, therefore, is

$$E_G\varphi \stackrel{\text{def}}{=} \bigwedge_{p_i \in G} K_i(p_i \in G \supset \varphi),$$

which says that $E_G\varphi$ holds iff each agent in G knows that, if it is a member of G, then φ holds. We choose this form of "everyone knows" as the appropriate form to generalize to indexical sets. Formally, we define $E_S\varphi$ by

$$E_S\varphi \stackrel{\text{def}}{=} \bigwedge_{p_i \in S} K_i(p_i \in S \supset \varphi).$$

We now define $C_S\varphi$ by

$$C_S\varphi \stackrel{\text{def}}{=} E_S\varphi \wedge E_S E_S\varphi \wedge \cdots \wedge E_S^m\varphi \wedge \cdots.$$

These definitions of E_S and C_S directly generalize the standard definitions from [HM84] and [DM86].

A useful tool for thinking about $E_S^k\varphi$ and $C_S\varphi$ is the *similarity graph* (*relative to* S). This is an undirected graph whose nodes are the points of the system, and whose edges are defined as follows: two points c and d are connected by an edge iff some agent p_i that is a member of both $S(c)$ and $S(d)$ has the same local state at both c and d (that is, $c \sim_i d$). For example, if S is the set $\mathcal{N}$ of nonfaulty processors, two points are connected by an edge in the similarity graph iff there is a processor that is nonfaulty at both points, and has the same local state at both points. The property of the similarity graph making is such a useful tool is that its connected components essential characterize what facts are common knowledge at any given point. To see this, we first note that an easy argument by induction on k shows that

Proposition 2.1: $c \models E_S^k\varphi$ iff $d \models \varphi$ for all points d of distance at most k from c in the similarity graph relative to S.

Proof: We proceed by induction on k. The induction hypothesis clearly holds for the case of $k = 0$ since $E_S^0\varphi = \varphi$ by definition.

Consider the case of $k = 1$. (Our previous restriction to nonempty indexical sets is crucial here.) Suppose $c \models E_S\varphi$. If d is of distance at most 1 from c, then some p_i in both $S(c)$ and $S(d)$ has the same local state at both c and d. Since $p_i \in S(c)$ and $c \models E_S\varphi$, we have $c \models K_i(p_i \in S \supset \varphi)$. Since $d \sim_i c$, we have $d \models (p_i \in S) \supset \varphi$; and since $p_i \in S(d)$, we have $d \models \varphi$. It follows that $d \models \varphi$ for all d of distance at most 1 from c. Suppose, conversely, that $d \models \varphi$ for all d of distance at most 1 from c. Suppose $p_i \in S(c)$, and suppose p_i has the same local state at both c and d. If $p_i \in S(d)$, then d is of distance at most 1 from c in the graph, so $d \models \varphi$ and hence $d \models (p_i \in S) \supset \varphi$. If $p_i \notin S(d)$, then clearly $d \models (p_i \in S) \supset \varphi$. Since this statement holds for all points $d \sim_i c$, we have $c \models K_i(p_i \in S \supset \varphi)$; and since this is true for all $p_i \in S(c)$, we have $c \models E_S\varphi$.

For $k > 1$, suppose the inductive hypothesis holds for $k - 1$. Notice that $c \models E_S^k\varphi$ iff $c \models E_S(E_S^{k-1}\varphi)$. By the induction hypothesis, $c \models E_S(E_S^{k-1}\varphi)$ iff $d \models E_S^{k-1}\varphi$ for all d of distance at most 1 from c, and $d \models E_S^{k-1}\varphi$ iff $e \models \varphi$ for all e of distance at most $k - 1$ from d. It follows that $c \models E_S^k\varphi$ iff $e \models \varphi$ for all e of distance at most k from c. □

Finally, since $c \models C_S\varphi$ iff $c \models E_S^k\varphi$ for all $k \geq 1$, it follows that

Proposition 2.2: $c \models C_S\varphi$ iff $d \models \varphi$ for all points d in c's connected component in the similarity graph relative to S.

Two points c and d are said to be *similar* (*relative to* $\mathcal{S}$), which we denote by $c \overset{\mathcal{S}}{\sim} d$, if they are in the same connected component of the similarity graph relative to $\mathcal{S}$. Since the indexical set $\mathcal{S}$ is generally clear from context (in Chapter 3 most often being the set $\mathcal{N}$ of nonfaulty processors), we denote similarity by $\sim$ without the superscript $\mathcal{S}$. We thus have:

Theorem 2.3: $c \models C_{\mathcal{S}}\varphi$ iff $d \models \varphi$ for all d satisfying $c \sim d$.

Our analysis in Chapter 3 will exploit this relationship between common knowledge and the similarity graph. The similarity graph will provide us with a useful combinatorial tool with which to study when facts become common knowledge.

2.3 Logic of Knowledge

We remark at this point that the definitions of knowledge and common knowledge we have given have been purely semantic definitions. We have talked about agents knowing facts, but we have not said where these facts come from other than to say that each fact corresponds to a set of points in a system. It is often convenient to have a formal, logical language of knowledge and common knowledge in which we can make statements about an agent's knowledge. We now show how to define such a language.

Let Φ be some arbitrary set of primitive propositions. Intuitively, these propositions are statements about points in the system that do not make explicit mention of an agent's knowledge, statements such as "*the value of register x is 0*" or "*processor p_i failed in round 3.*" Let $\mathcal{L}(\Phi)$ be the language obtained by closing Φ under the standard boolean connectives (conjunction and negation) and the knowledge operators of the form K_i, I_G, E_S, E_S^k, and C_S (one might also consider adding some of the standard modal operators from linear-time temporal logic such as $\Box$ and $\Diamond$). In other words, $\mathcal{L}(\Phi)$ is the smallest language with the property that if φ and ψ are contained in $\mathcal{L}(\Phi)$, then so are $\varphi \wedge \psi$, $\neg\varphi$, $K_i\varphi$, etc. Strings in the language $\mathcal{L}(\Phi)$ are called *formulas*. We use $\varphi \supset \psi$ as a shorthand for $\neg\varphi \vee \psi$, meaning that the truth of φ implies the truth of ψ.

So far, the formulas in $\mathcal{L}(\Phi)$ are just strings in a language with no intrinsic meaning in themselves. In order to give these formulas meaning in a system

$\mathcal{R}$, we require a truth assignment τ that maps each of the primitive propositions $\varphi \in \Phi$ to the set of points $\tau(\varphi)$ of $\mathcal{R}$ at which φ is true. Given such a truth assignment, the truth of an arbitrary formula $\varphi \in \mathcal{L}(\Phi)$ is defined by induction on the structure of φ using the definitions given above:

$$
\begin{array}{lll}
c \models \varphi & \text{iff} & c \in \tau(\varphi) \text{ whenever } \varphi \in \Phi \\
c \models \varphi \wedge \psi & \text{iff} & c \models \varphi \text{ and } c \models \psi \\
c \models \neg\varphi & \text{iff} & c \not\models \varphi \\
c \models K_i\varphi & \text{iff} & d \models \varphi \text{ for all } d \sim_i c \\
c \models I_G\varphi & \text{iff} & d \models \varphi \text{ for all } d \sim_G c \\
c \models E_S\varphi & \text{iff} & d \models \varphi \text{ whenever } d \sim_i c \text{ and } p_i \in \mathcal{S}(c)^1 \\
c \models E_S^k\varphi & \text{iff} & c \models E_S(E_S^{k-1}\varphi) \text{ whenever } k > 1 \\
c \models C_S\varphi & \text{iff} & c \models E_S^k\varphi \text{ for all } k \geq 1
\end{array}
$$

We assume that associated with every system $\mathcal{R}$ is a truth assignment $\tau_{\mathcal{R}}$ determining for every primitive proposition in Φ the points of $\mathcal{R}$ at which the proposition is true. From this assumption it follows that every formula in our language corresponds to the set of points of $\mathcal{R}$ at which the formula is true. Thus it follows that every formula in our language corresponds to a fact, a set of points of $\mathcal{R}$, as previously defined. For this reason we will sometimes abuse terminology and use the word "fact" in place of "formula." We remark that in later chapters we will be adding more knowledge operators to our language $\mathcal{L}(\Phi)$ as we refine our definitions of knowledge.

Finally, a formula φ is said to be *valid in the system* $\mathcal{R}$, which we denote by $\mathcal{R} \models \varphi$, if φ is true at all points of $\mathcal{R}$ (as determined by the system's truth assignment $\tau_{\mathcal{R}}$). A formula φ is said to be *valid*, which we denote by $\models \varphi$, if φ is valid in the system $\mathcal{R}$ for all systems $\mathcal{R}$.

2.4 Properties of Knowledge

The notions of knowledge, implicit knowledge, and common knowledge defined above are closely related to modal logics that have been extensively studied by philosophers (see [Hin62]). A modal operator M is said to *have*

[1]Notice that this is equivalent to defining $c \models E_s\varphi$ iff $c \models K_i(p_i \in \mathcal{S} \supset \varphi)$ for all $p_i \in \mathcal{S}(c)$. The advantage to our definition is we do not have to worry about whether $p_i \in \mathcal{S}$, and hence $p_i \in \mathcal{S} \supset \varphi$, is a formula in our language.

the properties of the modal system S5 if the following inference rule is satisfied for every system $\mathcal{R}$

1. if φ is valid in the system $\mathcal{R}$ then $M\varphi$ is valid in the system $\mathcal{R}$

and the following formulas are valid

2. $M\varphi \supset \varphi$,
3. $(M\varphi \wedge M(\varphi \supset \psi)) \supset M\psi$,
4. $M\varphi \supset MM\varphi$, and
5. $\neg M\varphi \supset M\neg M\varphi$.

If we take M to be the knowledge operator K_i, these statements may be interpreted as follows: the first statement says an agent knows all facts φ that are necessarily true; the second says an agent can know only true facts, since it says that if an agent knows φ then φ must be true; the third says an agent knows all consequences of its knowledge, since if it knows both φ and $\varphi \supset \psi$, then it also knows ψ; the fourth says that an agent knows what it knows, since if an agent knows φ, then it knows that it knows φ; and the fifth says that an agent knows what it doesn't know, since if an agent does *not* know φ, then it *knows* it does not know φ. It is not hard to show that the definitions of knowledge, implicit knowledge, and common knowledge as given above immediately implies the following (cf. [HM85, DM86]):

Proposition 2.4: The operators K_i, I_G, and C_s have the properties of the modal system S5.

Proof: We sketch the proof for the knowledge operator K_i, and leave the remaining operators for the reader.

1. Suppose φ is valid in the system $\mathcal{R}$. For any point c of $\mathcal{R}$, since φ is valid in the system it follows that $d \models \varphi$ for all points $d \sim_i c$, and hence that $c \models K_i\varphi$. Since $c \models K_i\varphi$ for any point c of $\mathcal{R}$, $K_i\varphi$ is valid in the system $\mathcal{R}$.

Let c be an arbitrary point of an arbitrary system $\mathcal{R}$.

2. If $c \models K_i\varphi$, then $d \models \varphi$ for all $d \sim_i c$, and in particular $c \models \varphi$.

3. If $c \models K_i\varphi$ and $c \models K_i(\varphi \supset \psi)$, then $d \models \varphi$ and $d \models \varphi \supset \psi$ for all $d \sim_i c$. It follows that $d \models \psi$ for all $d \sim_i c$, and hence that $c \models K_i\psi$.

4. Suppose $c \models K_i\varphi$, and suppose $d \sim_i c$. Notice that $e \sim_i d$ implies $e \sim_i c$, since $e \sim_i d$ and $d \sim_i c$ imply that p_i has the same local state at all three points. Since $c \models K_i\varphi$ implies $e \models \varphi$ for all $e \sim_i c$, it follows that $e \models \varphi$ for all $e \sim_i d$, and hence that $d \models K_i\varphi$. Since this is true for all $d \sim_i c$, it follows that $c \models K_iK_i\varphi$

5. Suppose $c \models \neg K_i\varphi$, and suppose $d \sim_i c$. Since $c \models \neg K_i\varphi$, we have $e \not\models \varphi$ for some $e \sim_i c$. But, as above, $e \sim_i d$ and hence $d \models \neg K_i\varphi$. Since this is true for all $d \sim_i c$, it follows that $c \models K_i\neg K_i\varphi$. □

In addition to the properties of S4, common knowledge satisfies two additional properties that will prove essential to our analysis in Chapter 3. One of these useful properties is the so-called *fixed point axiom*

$$C_s\varphi \equiv E_s(\varphi \wedge C_s\varphi).$$

or

$$C_s\varphi \equiv E_s(C_s\varphi)$$

which states that common knowledge is a fixed point of the E_s operator.[2] It implies that a fact's being common knowledge is in a sense "public:" a fact can be common knowledge to a group of agents only if all members of the group know that it is common knowledge. This axiom also implies that when a fact becomes common knowledge, it becomes common knowledge to all relevant agents simultaneously. The proof that common knowledge satisfies this fixed point axiom is instructive:

Proposition 2.5: The fixed point axiom "$C_s\varphi \equiv E_s(\varphi \wedge C_s\varphi)$" is valid.

Proof: Suppose $c \models C_s\varphi$ for some arbitrary point c of some arbitrary system $\mathcal{R}$. This means $c \models E_s^{k+1}\varphi$ for all $k \geq 0$. Since $E_s^{k+1}\varphi \equiv E_s(E_s^k\varphi)$, for every d adjacent to c we have $d \models E_s^k\varphi$ for all $k \geq 0$ by Proposition 2.1, and hence

[2]The two versions of the fixed point axiom turn out to be equivalent. The first version of the axiom generalizes more easily to variants of common knowledge considered in [HM84].

for every d adjacent to c we have both $d \models \varphi$ and $d \models C_s\varphi$ (remember that $E_s^0\varphi = \varphi$ by definition). It follows that $c \models E_s(\varphi \wedge C_s\varphi)$.

Suppose, conversely, $c \models E_s(\varphi \wedge C_s\varphi)$. By Proposition 2.1, this means $d \models \varphi \wedge C_s\varphi$ for all points d for distance at most 1 from c, and in particular that $c \models \varphi \wedge C_s\varphi$, so $c \models C_s\varphi$ as desired. □

The second useful property of common knowledge is captured by the following *induction rule*:

If	$\varphi \supset E_s\varphi$	is valid in the system $\mathcal{R}$,
then	$\varphi \supset C_s\varphi$	is valid in the system $\mathcal{R}$.

Roughly speaking, the induction rule implies that if a fact is "public" to a group of processors, in the sense that whenever it holds it is known to all members of the group, then whenever it holds it is in fact common knowledge.

Proposition 2.6: The induction rule "if $\varphi \supset E_s\varphi$ is valid in the system, then $\varphi \supset C_s\varphi$ is valid in the system" is sound.

Proof: Suppose $\varphi \supset E_s\varphi$ is valid in the system for some arbitrary system $\mathcal{R}$. To prove $\varphi \supset C_s\varphi$ is valid in the system $\mathcal{R}$, we assume $c \models \varphi$ and show $c \models C_s\varphi$. It is enough to show $c \models E_s^k\varphi$ for all $k \geq 0$. We proceed by induction on k. For $k = 0$, the fact that $c \models \varphi$ and that $E_s^0\varphi \equiv \varphi$ by definition imply $c \models E_s^0\varphi$. For $k > 0$, suppose the inductive hypothesis holds for $k - 1$. By the induction hypothesis for $k - 1$ we have $c \models E_s^{k-1}\varphi$, so Proposition 2.1 guarantees $d \models \varphi$ for all points d of distance at most $k - 1$. Since, however, $\varphi \supset E_s\varphi$ is valid in the system, we have $d \models E_s\varphi$, and Proposition 2.1 guarantees $e \models \varphi$ for all e of distance at most 1 from d. But this means $e \models \varphi$ for all e of distance at most k from c, and hence by Proposition 2.1 that $c \models E_s^k\varphi$ as desired. □

In the remainder of this work, the notions of knowledge, implicit knowledge, and common knowledge together with their properties proven in this section will be fundamental to our study of problems in distributed computing.

We end this chapter with a short discussion of the formulas or facts an agent is said to know. According to our definitions, facts are properties of points: they are either true or false at any given point. While facts are said to be true or false of points, many times the truth of a fact is determined by some simple property of a point. Many times, for example, the truth at

a point of a fact φ_1 like "the last coin flipped landed heads" is determined simply by the point's global state: given two points with the same global state, the fact is either true at both points or false at both points. Other times, the truth of a fact φ_2 like "all coins flipped in this run land heads" is determined simply by the run at the current point: given two points of the same run, the fact is either true at both points or false at both points, depending on whether all coins flipped in the run land heads. Notice that it is possible for a fact like φ_2 to be true at one point (r, k) and false at another point (r', k), even though they have the same global state. This is the case, for example, if all coins flipped in r land heads, all coins flipped in r' through time k land heads, and all coins flipped in r' after time k land tails.

Given a system $\mathcal{R}$, let us define a *property* to be a mapping from the points of $\mathcal{R}$ into some range; for example, mapping from a point (r, k) to the global state $r(k)$ or the run r. Intuitively, such a mapping maps a point to some property of the point that is of particular interest. Given a system $\mathcal{R}$ and a property P, we say a fact φ is a *fact about* P if fixing the value of P determines the truth of φ: given two points with the same value of P, the fact φ is either true at both points or false at both points. For example, if we assume the global state records the sequence of coins flipped so far in a run (perhaps this sequence is recorded in the environment), then the fact φ_1 above is a *fact about the global state* since the truth of φ_1 at two points with the same global state is the same; and φ_2 is a *fact about the run* since the truth of φ_2 at two points of the same run is the same.

Finally, recall our comment that the set Φ of primitive propositions in our language $\mathcal{L}(\Phi)$ typically consists of statements about the system that make no explicit mention of the agents' knowledge. In particular, it is common to take these propositions to be facts about the global state. In a given system $\mathcal{R}$, we say the language $\mathcal{L}(\Phi)$ is *state-generated* if each of the propositions $\varphi \in \Phi$ is a fact about the global state. This means the primitive propositions φ are simply statements about the global state (which we view as a particularly simple but fundamental kind of statement), and not, for example, about future events in the run.

Chapter 3

Programming Simultaneous Actions Using Common Knowledge

In this chapter, we show how thinking about distributed computation in terms of knowledge can aid in the design and analysis of protocols for a number of problems appearing in the literature, and in the proof of nontrivial lower bounds on the complexity of solving these problems in certain failure models.

3.1 Introduction

The problem of ensuring proper coordination between processors in distributed systems whose components are unreliable is both important and difficult. There are generally two aspects to such coordination: the actions the different processors perform, and the relative timing of these actions. Both aspects are crucial, for instance, in maintaining consistent views of a distributed database. In particular, it is often most desirable to perform coordinated actions *simultaneously* at different sites of a system. It is therefore of great interest to study the design of protocols involving simultaneous ac-

This chapter is joint work with Yoram Moses. Earlier versions have appeared in *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science* [MT86] and *Algorithmica* [MT88].

tions, actions performed simultaneously by all processors whenever they are performed at all.

In [DM86], Dwork and Moses study the design of protocols for simultaneous Byzantine agreement in the crash failure model, a failure model in which a processor fails by simply halting, never sending any message in any round following its halting round. Their analysis focuses on determining necessary and sufficient conditions for reaching simultaneous Byzantine agreement in terms of the processors' states of knowledge about the system. As a result of this analysis, they derive a protocol for simultaneous Byzantine agreement with the unique property of being *optimal in all runs*; that is, their protocol halts as early as any protocol for the problem could, given the pattern of faulty processor behavior that occurs. In contrast, previous protocols do not adapt their behavior on the basis of faulty processor behavior, and hence always perform as poorly as they do in their *worst case* run. Implicit in the work of Dwork and Moses is a general method for obtaining optimal protocols for many problems involving simultaneous actions in the crash failure model. Their technical analysis, however, makes strong use of particular properties of the crash failure model, and does not extend to more complicated failure models.

This chapter presents a novel approach to the design of fault-tolerant protocols in several variants of the more complex omissions failure model, a failure model in which processors fail only by intermittently failing to send some of the messages they are required by their protocol to send, but do not necessarily halt as in the crash model. We explicitly define a large class of *simultaneous choice problems*, a class intended to capture the essence of simultaneous coordination in synchronous systems. Many well-known problems, including simultaneous Byzantine agreement [PSL80, Fis83, DM86], distributed firing squad [BL87, CDDS85, Rab], etc., can be formulated as simultaneous choice problems. As the result of a delicate knowledge-based analysis in these failure models, we derive *at once* protocols that are *optimal in all runs* for all simultaneous choice problems: Each protocol is guaranteed to perform the desired simultaneous actions as soon as any protocol for the problem could, given the input to the system and the pattern of faulty processor behavior. (We will use *optimal* as shorthand for optimal in all runs.) Thus, we show how a knowledge-based analysis can be used as a general tool for the design of protocols for an entire class of problems. Our analysis applies to the crash failure model as well, and formally extends the statements

of results in [DM86] to the whole class of simultaneous choice problems (although most of the proof techniques we use are quite different from those in [DM86]).

Our approach is based on the close relationship between knowledge, communication, and action in distributed systems: A number of recent works (see [HM84], [DM86], and [Mos86]) show that simultaneous actions are closely related to common knowledge. Recall that, informally, a fact is *common knowledge* if it is true, everyone knows it, everyone knows that everyone knows it, and so on ad infinitum. Notice that every processor performing a simultaneous action knows the action is being performed. In addition, since such actions are performed simultaneously by all processors, every processor knows that all processors know the action is being performed. This argument can be (and will be) formalized and extended to show that when a simultaneous action is performed, it is common knowledge that the action is being performed. Consequently, a necessary condition for performing simultaneous actions is attaining common knowledge of particular facts (cf. [HF85]). Interestingly, our work shows that in a precise sense this is also a sufficient condition: The problem of performing simultaneous actions reduces to the problem of attaining common knowledge of particular facts.

In deriving optimal protocols for simultaneous choice problems, we make explicit and direct use of the relationship between common knowledge and simultaneous actions. The derivation proceeds in two stages. In the first stage, we program the optimal protocols in a high-level language where processors' actions depend on explicit tests for common knowledge of certain facts. These high-level protocols are extracted directly from the problem specifications via a few simple manipulations. The second stage deals with effectively implementing these tests for common knowledge. We give a direct implementation of such tests in all variants of the omissions failure model we consider. As a result, our high-level protocols have effective implementations in these failure models as low-level, standard protocols that are optimal in all runs.

Consider, for example, the following version of the *distributed firing squad* problem (cf. [BL87, CDDS85, Rab]): An external source may send "start" signals to some of the processors in the system at unpredictable times, possibly different times for different processors. It is required that (i) if any nonfaulty processor receives a "start" signal, then all nonfaulty processors perform an irreversible "firing" action at some later point (which means each

nonfaulty processor enters some distinguished "firing" state it never leaves), (ii) whenever any nonfaulty processor "fires," all nonfaulty processors do so simultaneously, and (iii) if no processor receives a "start" signal, then no nonfaulty processor "fires." The high-level protocol we derive for this problem in the omissions model requires all processors to act as follows:

repeat *every round*
  send current local state to every processor
until *it is* **common knowledge** *that*
  some processor received a "start" signal;
"fire" and **halt.**

Since we exhibit an effective implementation of the test for common knowledge embedded in this protocol, this high-level protocol can be transformed into a standard protocol that is optimal in all runs. No previous protocol for this problem suggested in the literature is optimal in all runs. Furthermore, in many cases this protocol "fires" much earlier than any other known protocol for this problem: In some cases, this protocol "fires" as soon as one round after the first "start" signal is received.

We show that optimal protocols for simultaneous choice problems can always be implemented in a communication efficient way, in all variants of the omissions model we consider. However, our direct implementation of tests for common knowledge is not *computationally* efficient: It requires processors to perform exponential-time computations between consecutive rounds of communication. One of the major technical contributions of this chapter is a method of efficiently implementing tests for common knowledge in several variants of the omissions failure model. In the standard omissions model, a failure model in which processors fail only by intermittently failing to send some of the messages they are required by their protocol to send, we provide a clean and concise method of efficiently implementing tests for common knowledge. The analysis underlying this method reveals the basic combinatorial structure underlying the omissions model, as well as crisply characterizing the set of facts that can be common knowledge at any point in the execution of a protocol. In the *receiving omissions* model, a failure model in which processors fail only by intermittently failing to *receive* some of the messages sent to them rather than failing to *send* messages, testing for common knowledge is shown to be trivial. This exposes a significant difference between two seemingly symmetric failure models.

We are *not* able to efficiently implement tests for common knowledge in the *generalized omissions* model, in which faulty processors may fail both to send *and* to receive messages. In fact, we show that testing for common knowledge in this model is NP-hard. As a result, using the close relationship between common knowledge and simultaneous actions, we are able to show that no optimal protocol for any reasonable simultaneous choice problem can be computationally efficient unless P=NP. In particular, in this model there can be no computationally-efficient optimal protocol for the distributed firing squad problem stated above, for simultaneously performing Byzantine agreement (see [PSL80, DM86]), or for most any other simultaneous problem. We consider another variant of the omissions model, called *generalized omissions with information*, in which it is assumed that the intended receiver of an undelivered message can test (and therefore knows) whether it or the sender is at fault. We show that the techniques used in the standard omissions model extend to this model as well, yielding computationally-efficient optimal protocols. As a result, we see that optimal protocols for simultaneous choice problems are computationally intractable in the generalized omissions model precisely because of the fact that in this model undelivered messages do not uniquely determine the set of faulty processors.

Thus, we show how to derive efficient optimal protocols in the omissions model, and we show that optimal protocols are intractable in the generalized omissions model. Since it is unrealistic to expect conventional processors (limited to polynomial-time computation) to follow such intractable protocols, it becomes becomes interesting to ask how well resource-bounded processors can perform simultaneous actions in the generalized omissions model. Analyzing this problem requires extending the theory of knowledge given in Chapter 2 to account for the restricted computational power of such processors. Such an extension should give rise to notions of resource-bounded knowledge and common knowledge that closely correspond to the ability of resource-bounded processors to perform simultaneous actions. The need for a theory of resource-bounded knowledge has already been demonstrated, primarily by cryptographic problems (e.g., [GM84, GMR89]), in which computational complexity is introduced artificially by restricting the computational power of the adversary, thus allowing solutions involving encryption. This work, however, provides a more compelling indication of the need for such a theory, even for the analysis of simple problems in distributed computation that do not make such assumptions about the adversary. We note that some

such notions of knowledge have since been proposed [Mos88, HMT88, FZ88], and we will return to the need for such notions in Chapter 5 when we study cryptographic protocols in terms of knowledge.

Since some of the proofs in this chapter are quite technical, their details can make it difficult to obtain a high-level understanding of this work. We strongly recommend that the reader skip all proofs on the first reading. The rest of this chapter is organized as follows: Section 3.2 defines the model of distributed systems used in the chapter. In Section 3.3 we define the notion of a *simultaneous choice problem*, a large class of problems involving coordinated simultaneous actions. Section 3.4 presents a uniform method of deriving an optimal high-level protocol from the specification of a simultaneous choice problem, using explicit tests for common knowledge. Section 3.5 deals with the problem of efficiently implementing tests for common knowledge of facts relevant to simultaneous choice problems in a number of failure models. This section is the heart of the chapter. The analysis in this section reveals interesting properties of the different failure models, and exposes fine distinctions between them. Finally, Section 3.6 contains some concluding remarks.

3.2 Model of a System

This section introduces a model of the distributed systems with which this chapter is concerned, an elaboration of the model given in Chapter 2. Our treatment extends and is closely related to that of [DM86].

We consider synchronous systems of unreliable processors. Such a system consists of a finite collection $P = \{p_1, \ldots, p_n\}$ of $n \geq 2$ processors, each pair of which is connected by a two-way communication link, and each sharing a common global clock that starts at time 0 and advances in increments of one.[1] We model such systems by elaborating the model of computation given in Chapter 2 in the following ways. In addition to receiving messages from other processors at the end of a round, a processor may also receive requests for service from *clients* external to the system (think, for example, of a distributed airline reservation system). These external requests from the

[1] We assume the existence of a shared global clock for ease of exposition. The analysis performed in this chapter applies even if the processors have their own local clocks, possibly displaying different times, as long as the clocks tick (or advance) at the same rate.

clients are considered distinct from the internal messages sent by processors in the system. Actions resulting from the servicing of such requests may take a variety of forms, including the initiation of various activities within the system by sending certain messages to other processors in later rounds. Each message sent by a processor is assumed to include the identities of the sender and intended receiver of the message, as well as the round in which it is sent; similarly for each request. At any given time, a processor's *message history* is a set containing the messages it has received so far from the other processors, and a processor's *input history* is a set containing its initial state together with the requests it has received so far from the system's external clients. A processor's *local state* at any given time consists of its message history, its input history, the time on the global clock, and the processor's identity. For technical reasons, it will be convenient to talk about processors' states at negative times (before time 0). A processor's state at a negative time is defined to be a distinguished *empty* state.

We assume processors are following a *deterministic protocol* as defined in Chapter 2. Notice, however, that the state protocol component of a processor's local protocol is no longer of interest since we have already described how a processor's local state should change from round to round, and we will ignore it for the remainder of this chapter. Consequently, an equivalent definition of a protocol is a function from processor's local state to a list of actions the processor is required to perform, followed by a list of messages the processor is required to send. While we assume that all processors in the system faithfully follow their protocols, sending and receiving messages as required, some messages may be lost due to failures in the system. A run of a protocol in the absence of any such failure is defined precisely as defined in Chapter 2. In the presence of failures, however, we must elaborate this definition: given a run in which failures occur, a processor's message history at time k no longer records all messages sent to it during round k since some of these messages may be lost. (Of course, the processor's message history at time k *will* record all messages recorded in its message history at time $k-1$.) We attribute lost messages to failures on the part of processors (due to the failures of their input or output ports, say), and the various failure models we consider differ only in how we assign these failures to processors. We consider the following *failure models*:

- the *omissions* model ([MSF83]), in which a lost message indicates that

the sender of the message is faulty;

- the *receiving omissions* model, in which a lost message indicates that the receiver is faulty;
- the *generalized omissions* model ([PT86]), in which a lost message indicates that either the sender *or* receiver is faulty; and
- *generalized omissions with information*, which differs from the generalized omissions model in that the intended receiver of a lost message is told whether the sender or the receiver is faulty.

When the sender of a lost message is said to be at fault, we say the processor failed to send the message; and when the receiver of a lost message is said to be at fault, we say the processor failed to receive the message.

We now define the notion of a failure pattern, a formal description of faulty processor behavior during a run. The notion of a failure pattern in each variant of the omissions model is a suitable restriction of the general definition given here. Remember that a faulty processor may fail to send or receive certain messages. It is therefore natural to define the *faulty behavior* of a processor p to be a pair of functions S and R mapping round numbers to sets of processors. Intuitively, these are the processors p fails to send messages to or receive messages from, respectively, during each round. A *failure pattern* is a collection of faulty behaviors $\langle S_i, R_i \rangle$, one for each processor p_i. The processor p_i is said to be *faulty* in such a failure pattern if either of the sets $S_i(k)$ or $R_i(k)$ is nonempty for some k, in which case p_i is said to *fail* during round k, and p_i is said to be *nonfaulty* otherwise. If, for example, the set $S_i(k)$ contains the processor p_j, we say that p_i is faulty since any message p_i's protocol requires that it send to p_j will be lost. Notice, however, that a faulty processor need not actually *exhibit* any faulty behavior at all since the fact that any message from p_i to p_j during round k is lost will never be discovered if p_i's protocol does not require it to send any message to p_j in round k.

The *failure pattern* of a run is a failure pattern with the property that in every round k each processor p_i sends no messages to processors in $S_i(k)$ but sends all required messages to processors not in $S_i(k)$, and receives no messages from processors in $R_i(k)$ but receives all messages sent to it by processors not in $R_i(k)$. Notice, by the way, that a run may be consistent

with more than one failure pattern if the protocol being followed does not require processors to send messages to every processor in every round. Given a run r, if γ_i is the complete input history of processor p_i in r, then we say that $\gamma = (\gamma_1, \ldots, \gamma_n)$ is the *input* to r.

A pair (π, γ), where π is a failure pattern and γ is an input, is called an *operating environment.* Notice that an operating environment is independent of any particular protocol. An operating environment simply determines for each processor and for each round what faulty behavior it will exhibit (if any) during the round and what external requests it will receive during the round, regardless of the protocol the processor is following. Given an operating environment together with a particular protocol, however, the two uniquely determine a run of the given protocol (in the given operating environment). Two runs of two different protocols are said to be *corresponding runs* if they have the same operating environment. The fact that an operating environment is independent of the protocol will allow us to compare different protocols according to their behavior in corresponding runs.

In many systems of interest, the environment reacts to the protocol being followed by the system, meaning that the input the system received from the environment can depend on the output to the environment generated by the system. One can imagine, for example, a bank customer walking up to a teller to withdraw \$100. If the teller's "protocol" causes the teller to hand the customer 100 one dollar bills, the customer will probably ask for two \$50 bills instead. If the teller's "protocol" causes the teller to hand the customer a single \$100 bill, the customer may not ask for two \$50 bills. Because the environment reacts differently to the two teller protocols, making different requests in the context of the different protocols, it seems difficult to compare the two protocols in the context of a fixed sequence of requests by the bank customer. In contrast, however, we are interested in protocols that react to their environment, and not the environment's reaction to the protocol. Our method of comparing protocols does not allow us to study the interaction of protocols and their environment from both points of view.

In this work, we study the behavior of protocols in the presence of a bounded number of failures (of a particular type) and a given setting of possible inputs. It is therefore natural to identify a *system* with the set of all possible runs of a given protocol under such circumstances. Formally, a system is identified with the set of runs of a protocol $\mathcal{P}$ with $n \geq 2$ processors of which at most $t \leq n-2$ may be faulty (in the sense of a particular

failure model $\mathcal{M}$ defined above), where the complete input history of each processor p_i is an element of a set Γ_i. We denote this set of runs by the tuple $\Sigma = (n, t, \mathcal{P}, \mathcal{M}, \Gamma_1, \ldots, \Gamma_n)$. Our definition of a system ensures that the input to the system is orthogonal to, and hence carries no information about, the failure pattern. In addition, since the set of possible inputs in the system has the form $\Gamma_1 \times \cdots \times \Gamma_n$, one processor's input contains no information about any other processor's input, and hence the only way in which processors obtain information about other processors' input is via messages communicated between the processors in the system.

While a protocol may be thought of as a function of processors' states, protocols for distributed systems (as well as protocols for sequential and parallel computation) are typically written uniformly in terms of the number n of processors and the number t of failures tolerated, for values of n and t of virtually arbitrary size (although requirements such as $n > 2t$ must sometimes be satisfied in order for the protocol to behave correctly). In this sense, the protocol is parameterized by n and t, and the actions and messages required of a processor by a protocol may be viewed as depending on n and t as well as the processor's state. Therefore, for the purposes of this chapter, we assume that a *protocol* is a function from n, t, and a processor's local state to a list of actions the processor is required to perform, followed by a list of messages the processor is required to send.[2] Since each protocol is defined for systems of arbitrary size, it is natural to define a *class* of systems to be a collection of systems $\{\Sigma(n,t) : n \geq t + 2 \geq 0\}$, where $\Sigma(n,t) = (n, t, \mathcal{P}, \mathcal{M}, \Gamma_1, \ldots, \Gamma_n)$ for some fixed protocol $\mathcal{P}$, failure model $\mathcal{M}$, and input sets Γ_i.

[2]Notice that processors must compute this function by following some algorithm. Thus, while we formally define a protocol in terms of functions, it is convenient to maintain both views of a protocol as a function and an algorithm.

3.3 Simultaneous Choice Problems

In this section we define the class of *simultaneous choice problems* for which we construct optimal protocols, a large class of problems that capture the essence of coordinated simultaneous action in a distributed environment. Roughly speaking, these problems require that one of a number of alternative actions be performed (or "chosen") simultaneously by the nonfaulty processors, where for each action we are given conditions under which the action *must* be performed and conditions under which its performance is forbidden. In addition to these conditions, the specification of such a problem must also determine the possible operating environments in which such a choice is to be made, by specifying what inputs each processor may possibly receive and what types of processor failures are possible.

We think of an *action* as something special that can be done by a processor. An action might be writing the value 1 to an output register, or entering some distinguished state such as the "firing" state in the distributed firing squad problem. Formally, an action can be modeled as a message a processor can send to the environment. There is nothing about the action itself that restricts its performance, say, to time k but not to time $k+1$. A *simultaneous action* a is an action with two associated conditions $pro(a)$ and $con(a)$ stating when the action a should or should not be performed. Recall that a run is determined by a protocol and an operating environment; it follows that the operating environment is the most general protocol-independent aspect of a run a problem specification can refer to when stating when an action should or should not be performed. Consequently, we assume both $pro(a)$ and $con(a)$ are facts about the operating environment.

A *simultaneous choice problem* (or simply a *simultaneous choice*) C is determined by a set $\{a_1, \ldots, a_m\}$ of simultaneous actions and their associated conditions, together with a failure model $\mathcal{M}$, and a set Γ_j of complete input histories for each processor p_j. Intuitively, we want all of the nonfaulty processors to choose one of the actions a_i that they can perform without violating the $pro(a_j)$ and $con(a_j)$ conditions, and to perform a_i simultaneously. Since the $pro(a_j)$ and $con(a_j)$ conditions are facts about the operating environment, which means they depend on the input and failure patterns, we include in the problem specification the sets Γ_j determining the possible input patterns and the failure model $\mathcal{M}$ determining the possible failure patterns. ($\mathcal{M}$ will always be one of the failure models defined in Section 3.2.)

Loosely speaking, we want every run r of a protocol implementing $\mathcal{C}$ satisfy the following conditions:

(i) each nonfaulty processor performs at most one of the a_i's,

(ii) any a_i performed by some nonfaulty[3] processor is performed *simultaneously* by all of them,

(iii) a_i is performed by all nonfaulty processors if r satisfies $pro(a_i)$, and

(iv) a_i is *not* performed by any nonfaulty processor if r satisfies $con(a_i)$.

More formally, a protocol $\mathcal{P}$ and the simultaneous choice $\mathcal{C}$ *determine* a class of systems $\{\Sigma(n,t) : n \geq t+2\}$, where $\Sigma(n,t) = (n,t,\mathcal{P},\mathcal{M},\Gamma_1,\ldots,\Gamma_n)$. We say that $\mathcal{P}$ *implements* $\mathcal{C}$ if every run of every system in the class determined by $\mathcal{P}$ and $\mathcal{C}$ satisfies the conditions (i)-(iv) above. A simultaneous choice is said to be *implementable* (or *satisfiable*) if there is a protocol that implements it. We note that both $\mathcal{P}$ and $\mathcal{C}$ are required to completely determine a system (a set of runs): because a run is determined by a protocol and an operating environment, the protocol $\mathcal{P}$ is clearly required, and the failure model $\mathcal{M}$ and input sets Γ_i contributed by $\mathcal{C}$ are required to determine the set of possible operating environments.

This definition of a simultaneous choice is fairly abstract. However, many familiar problems requiring simultaneous action by a group of processors are instances of a simultaneous choice. In all known cases, the conditions $pro(a_i)$ and $con(a_i)$ are facts about the input and the existence of failures, and hence are facts about the operating environment. (By *the existence of failures* we mean whether *any failure whatsoever* occurs during the run. Some problems allow the nonfaulty processors to display default behavior in the presence of failures; see [LF82].) For example, the distributed firing squad problem is a simultaneous choice consisting of a single "firing" action a, with the condition $pro(a)$ being the receipt of a "start" signal by a nonfaulty processor, and the condition $con(a)$ being that no processor receives a "start" signal. Each set Γ_j

[3]We have chosen the set $\mathcal{N}$ of nonfaulty processors as the set of processors required to perform actions simultaneously, but the notion of a simultaneous choice problem may be stated in terms of many other similar (indexical) sets of processors, including the set P of all processors, with the analysis in this section and the next one carrying through without change.

of possible inputs simply allows for a "start" message to be delivered to any processor at any time.

In addition to simultaneous choice problems, we also consider the closely related class of *strict* simultaneous choice problems. Both classes are specified in essentially the same way, except that runs of a protocol implementing a *strict* simultaneous choice are required to satisfy the modified condition

(i′) each nonfaulty processor performs *exactly* one of the a_i's,

together with conditions (ii)-(iv) above. All of the results in this chapter hold for a strict simultaneous choice as well as a simultaneous choice, and henceforth we will mention explicitly only to a simultaneous choice.

The simultaneous Byzantine agreement problem (see [DM86, PSL80]) is an example of a strict simultaneous choice. This problem consists of an action a_0 of "deciding 0" and an action a_1 of "deciding 1." Each set Γ_j of possible inputs consists of two possible inputs: one starting with initial value 0 and receiving no further external input during the run, and the other starting with initial value 1. The condition $pro(a_0)$ is that all initial values are 0, and the condition $pro(a_1)$ is that all initial values are 1. The conditions $con(a_0)$ and $con(a_1)$ are both taken to be *false*. Simultaneous Byzantine agreement is a *strict* simultaneous choice, since the processors are required to decide either 0 or 1 in every run. Other related problems that may also be formulated as (strict) simultaneous choice problems include weak Byzantine agreement and the Byzantine Generals problem (see [Fis83]).

Having formally defined a simultaneous choice (and a *strict* simultaneous choice), let us consider when the specification of such a problem disallows performing a simultaneous action a_i. Clearly, if $con(a_i)$ holds then performing a_i is disallowed. In addition, since by condition (i) no more than one action may be performed by the nonfaulty processors in any given run, the condition $pro(a_j)$, for some $j \neq i$, requires a_j to be performed, and hence also disallows a_i. It is easy to see that these are the only conditions under which performing a_i is disallowed. This motivates the following definition:

$$enabled(a_i) \stackrel{\text{def}}{=} \neg con(a_i) \wedge \bigwedge_{j \neq i} \neg pro(a_j).$$

Our discussion above implies that the performance of an action a_i is allowed by the problem specification iff the condition $enabled(a_i)$ is satisfied. Notice that it is possible for several of the conditions $enabled(a_i)$ to hold at

once, in which case performance of any of the enabled actions is allowed by the problem specification. In addition, it is easy to see that the formulas $con(a_i) \supset \neg enabled(a_i)$ and $pro(a_i) \supset \neg enabled(a_j)$ $(j \neq i)$ are valid in any system in which processors follow a protocol implementing a simultaneous choice. Finally, notice that because the conditions $pro(a_j)$ and $con(a_j)$ are facts about the operating environment, so is each condition $enabled(a_i)$.

As an example, notice that the condition $enabled(a)$ for the distributed firing squad problem is simply that *some* processor receives a "start" signal. For the simultaneous Byzantine agreement problem, the condition $enabled(a_0)$ is that some initial value is 0, and the condition $enabled(a_1)$ is that some initial value is 1. Since for most assignments of initial values both $enabled(a_0)$ and $enabled(a_1)$ hold, it is typically the case that deciding either 0 or 1 is acceptable. It need not be the case (and, in fact, usually will *not* be the case) that the conditions $enabled(a_i)$ for a typical simultaneous choice will be mutually exclusive.

Having formally defined the notion of a simultaneous action, we are now in a position to carefully state the relationship between simultaneous actions and common knowledge mentioned in the introduction: When a simultaneous action is performed, it is common knowledge that the action is being performed. The statement we actually prove is that when such an action is performed, it is common knowledge that the action is enabled. This is the first (and the key) relationship we establish between common knowledge and the performance of simultaneous actions.

Lemma 3.1: Let r be a run of a protocol implementing a simultaneous choice C. If the action a_i of C is performed by a nonfaulty processor at time ℓ in r, then $(r, \ell) \models C_N enabled(a_i)$.

Proof: Let φ be the fact "a_i is being performed by a nonfaulty processor." A processor p_j performing the action a_i clearly knows that it is performing a_i. This processor therefore also knows that if it is nonfaulty, then a_i is being performed by a nonfaulty processor. Since r is a run of a protocol implementing C, the action a_i is performed simultaneously by all nonfaulty processors whenever it is performed by a single nonfaulty processor. It follows that whenever φ holds, so does $E_N\varphi$, and hence $\varphi \supset E_N\varphi$ is valid in the system. The induction rule implies that $\varphi \supset C_N\varphi$ is valid in the system as well. Notice that $\varphi \supset enabled(a_i)$ is valid in the system. It

follows that $C_N\varphi \supset C_N enabled(a_i)$ is valid in the system, and hence so is $\varphi \supset C_N enabled(a_i)$. Thus, $(r, \ell) \models \varphi$ implies $(r, \ell) \models C_N enabled(a_i)$, and we are done. □

In the above proof, the essential fact that $\varphi \supset E_N\varphi$ is valid in the system depends crucially on our definition of $E_N\varphi$. As discussed in Chapter 2, a processor p performing a_i knows that a_i is being performed, but since a nonfaulty processor might not know that it is nonfaulty, p might *not* know that a_i is being performed by a nonfaulty processor. The processor p *does* know, however, that if *it* (p itself) is nonfaulty, then a nonfaulty processor is performing a_i. It is for this reason that we have been led to choose our definition of $E_N\varphi$ as we have, as discussed in Chapter 2.

3.4 Optimal Protocols

In this section, we show how to extract a high-level optimal protocol for a simultaneous choice problem directly from its specification. (As mentioned in the introduction, we use the word *optimal* as shorthand for optimal in all runs; recall that this optimality is in terms of the number of rounds required to perform a simultaneous choice.) We begin by considering a simple class of protocols that will serve as a building block in the design of such optimal protocols. Recall that we think of a protocol as having two components, an *action* protocol and a *message* protocol. A protocol is said to be a *full-information protocol* (cf. [Had83, FL82, PSL80]) if its message protocol is:

repeat every round
    send current local state to all processors
forever.

Intuitively, since such a protocol requires that all processors send all of the information available to them in every round, one would expect this protocol to give each processor as much information about the operating environment as any protocol could. In particular, the following result shows that if a processor cannot distinguish two operating environments during runs of a full-information protocol, then the processor cannot distinguish these operating environments during runs of any other protocol.

Lemma 3.2: Let r and r' be runs of a full-information protocol $\mathcal{F}$, and let s and s' be runs of an arbitrary protocol $\mathcal{P}$ corresponding to r and r', respectively. For all processors q and times ℓ, if $r_q(\ell) = r'_q(\ell)$ then $s_q(\ell) = s'_q(\ell)$.

Proof: We proceed by induction on the time ℓ. The case of $\ell = 0$ is immediate since q must have the same initial state in both r and r', and hence also in s and s'. Suppose $\ell > 0$ and the inductive hypothesis holds for all processors p at time $\ell - 1$. The local state of q at time ℓ is determined by its local state at time $\ell - 1$, the (external) input it receives during round ℓ, and the messages it receives during round ℓ. Since q has the same local state at time $\ell - 1$ in r and r', by the inductive hypothesis, the same is true in s and s'. Since q receives the same input during round ℓ in r and r', the same is true in s and s'. If q does not receive a message from p during round ℓ in r and r', then both operating environments determine that no message from p to q during round ℓ is delivered. Thus, q does not receive a message from p during round ℓ in either s or s'. If q *does* receive a message from p during round ℓ in r and r', then both operating environments determine that any message from p to q during round ℓ is delivered. If q receives a message from p during round ℓ of r and r', then since q must receive the same message from p in both r and r', the local state of p must be the same at time $\ell - 1$ in r and r'. By the inductive hypothesis, p's local state at time $\ell - 1$ must also be the same in s and s'. Since $\mathcal{P}$ is a deterministic function of processor states, q receives the same messages from p during round ℓ in s and s'. Thus, q has the same local state at time ℓ in s and s'. □

Thus, roughly speaking, processors learn the most about the operating environment during runs of full-information protocols. The following corollary of Lemma 3.2 shows that facts about the operating environment become common knowledge during runs of such protocols at least as soon as they do during runs of any other protocol. This result captures in a precise sense a property of full-information protocols that is essential to our analysis.

Corollary 3.3: Let φ be a fact about the operating environment. Let r and s be corresponding runs of a full-information protocol $\mathcal{F}$ and an arbitrary protocol $\mathcal{P}$, respectively. If $(s, \ell) \models C_N\varphi$ then $(r, \ell) \models C_N\varphi$.

Proof: Suppose that $(s, \ell) \models C_N\varphi$. We will prove that $(r, \ell) \models C_N\varphi$ by showing that $(r', \ell) \models \varphi$ for all runs r' of $\mathcal{F}$ such that $(r, \ell) \sim (r', \ell)$; that is,

that (r, ℓ) and (r', ℓ) are in the same connected component of the similarity graph. Fix r', and let s' be the run of $\mathcal{P}$ corresponding to r'. Lemma 3.2 and a simple inductive argument on the distance between (r, ℓ) and (r', ℓ) in the similarity graph show that $(r, \ell) \sim (r', \ell)$ implies $(s, \ell) \sim (s', \ell)$. Since $(s, \ell) \models C_N\varphi$, we have $(s', \ell) \models \varphi$. Since corresponding runs satisfy the same facts about the operating environment, $(s', \ell) \models \varphi$ implies $(r', \ell) \models \varphi$. It follows that $(r, \ell) \models C_N\varphi$. □

We are now in a position to describe how to construct optimal protocols for simultaneous choice problems. Recall that when a simultaneous action a_i is performed, Lemma 3.1 implies that *enabled*(a_i) must be common knowledge. Since *enabled*(a_i) is a fact about the operating environment, Corollary 3.3 implies that *enabled*(a_i) becomes common knowledge in runs of a full-information protocol as soon at it does in corresponding runs of any other protocol. Thus, given an effective test that the nonfaulty processors can use to determine whether *enabled*(a_i) is common knowledge, a test we denote by *test-for-*C_N*enabled*(a_i), the following protocol $\mathcal{F}_c$ is an optimal protocol for C:

```
no_action_performed ← true;
repeat every round
   if no_action_performed and
                 test-for-C_N enabled(a_i) returns true for some a_i
      then
         j ← min {i : test-for-C_N enabled(a_i) returns true},
         perform a_j,
         no_action_performed ← false;
   send current local state to every processor;
forever.
```

Before formally proving that $\mathcal{F}_c$ is an optimal protocol, we must define more formally the tests for common knowledge that appear in $\mathcal{F}_c$. Recall that the fixpoint axiom implies that $C_N\varphi \supset E_N C_N\varphi$ is valid. This guarantees that $C_N\varphi$ follows from the local state of each nonfaulty processor whenever $C_N\varphi$ holds. In other words, since $C_N\varphi$ implies $E_N C_N\varphi$ which, for every nonfaulty processor p_i, implies $K_i C_N\varphi$, each nonfaulty processor can determined from its local state that $C_N\varphi$ holds. This is *not* true for faulty processors.

It is therefore natural to define a *test for common knowledge of* φ, denoted as above by *test-for-*$C_N\varphi$, to be a test that, given the local state of a nonfaulty

processor at (r, ℓ) (together with n and t), returns **true** iff $C_{\mathcal{N}}\varphi$ holds at (r, ℓ). Such a test may return either **true** or **false** when given the local state of a faulty processor. Let us denote by $\mathcal{A}_j(r, \ell)$ the set of actions a_i such that *test-for-*$C_{\mathcal{N}}$*enabled*(a_i) returns **true** when given the local state of p_j at (r, ℓ). Notice that if p_j is nonfaulty, then $\mathcal{A}_j(r, \ell)$ is precisely the set of actions a_i such that $C_{\mathcal{N}}$*enabled*(a_i) holds at (r, ℓ). It follows that for all nonfaulty processors p_j the sets $\mathcal{A}_j$ are equal at all times. In particular, all become nonempty at the same time (as soon as *enabled*(a_i) becomes common knowledge for some a_i). Thus, if all processors p_j choose the action of least index from $\mathcal{A}_j$ as soon as this set becomes nonempty, as required by $\mathcal{F}_c$, then all nonfaulty processors choose the same action simultaneously. We can now prove that $\mathcal{F}_c$ is an optimal protocol for $\mathcal{C}$. (Recall that a simultaneous choice problem is *implementable* iff there exists a protocol that implements it.)

Theorem 3.4: If $\mathcal{C}$ is an implementable simultaneous choice problem, then $\mathcal{F}_c$ is an optimal protocol for $\mathcal{C}$.

Proof: We first prove that nonfaulty processors perform actions in runs of $\mathcal{F}_c$ as soon as they do in corresponding runs of any protocol implementing $\mathcal{C}$. Let r be a run of $\mathcal{F}_c$, and let s be the corresponding run of a protocol implementing $\mathcal{C}$. Lemma 3.1 implies that if a_i is performed by a nonfaulty processor at time ℓ in s, then $(s, \ell) \models C_{\mathcal{N}}$*enabled*$(a_i)$. Since *enabled*$(a_i)$ is a fact about the operating environment, Corollary 3.3 implies that $(r, \ell) \models C_{\mathcal{N}}$*enabled*$(a_i)$. As a result, $\mathcal{A}_j(r, \ell)$ must be nonempty for all nonfaulty processors p_j, and hence each must perform an action in r no later that time ℓ. It follows that nonfaulty processors perform actions in runs of $\mathcal{F}_c$ as soon as they do in corresponding runs of any protocol implementing $\mathcal{C}$.

We now show that $\mathcal{F}_c$ actually implements $\mathcal{C}$. Let r be a run of $\mathcal{F}_c$. First, it is obvious from the definition of $\mathcal{F}_c$ that each nonfaulty processor performs at most one action in r. (If $\mathcal{C}$ is an implementable *strict* simultaneous choice, then the preceding discussion shows that the nonfaulty processors perform *exactly* one action in r.) Second, if a nonfaulty processor p_j performs an action a_i at time ℓ during r, then time ℓ is the first time at which $\mathcal{A}_j(r, k)$ is nonempty, and a_i is the action of least index in this set. Since $\mathcal{A}_j(r, k) = \mathcal{A}_m(r, k)$ for all nonfaulty processors p_m, the same is true for all nonfaulty processors. As a result, all nonfaulty processors must choose to perform a_i

simultaneously at time ℓ. Third, if r satisfies $pro(a_i)$, then the run s of any protocol implementing C corresponding tc r must satisfy $pro(a_i)$, and hence a_i must be performed in s. As we have already seen, an action must also be performed in r. Since $pro(a_i) \supset \neg enabled(a_j)$ for all $j \neq i$, the set $\mathcal{A}_j(r,k)$ of a nonfaulty processor p_j must contain no action other than a_i (if it contains any action at all). Thus, a_i must be the action performed in r. Finally, if r satisfies $con(a_i)$, then r does not satisfy $enabled(a_i)$, and no set $\mathcal{A}_j(r,\ell)$ for any nonfaulty processor p_j contains a_i. Thus, a_i is not performed in r. It follows that $\mathcal{F}_c$ implements C. □

As a result of Theorem 3.4, we see that full-information protocols can be used as the basis of optimal protocols for simultaneous choice problems. Thus, we will restrict our attention to full-information protocols in the remainder of this chapter: Unless otherwise specified, all protocols mentioned will be full-information protocols, and all runs will be runs of such protocols. More important, however, a consequence of Theorem 3.4 is that designing an optimal protocol for a simultaneous choice problem C essentially reduces to testing for common knowledge of certain facts: In order to design an optimal protocol for C, it is enough to construct the tests for common knowledge of the facts $enabled(a_i)$. We note that the fundamental property of common knowledge underlying the existence of such tests is the fact that $C_N\varphi \supset E_N C_N \varphi$ is valid; that is, when φ becomes common knowledge, the fact that φ is common knowledge will follow from the local state of every nonfaulty processor. The problem of implementing such tests is the subject of the following section.

Before ending this section, however, we consider the size of messages required by a full-information protocol $\mathcal{F}$. Such a protocol requires processors to send their entire local state during every round. Since, strictly speaking, the size of a processor's state may be exponential in the number of rounds elapsed, this protocol seems to require processors to send messages of exponential length. We now show, however, that in the variants of the omissions model we consider in this work there is a simple, compact representation of a processor's state that may be sent instead. Consequently, it will be possible to implement all full-information protocols (and in particular the optimal protocol $\mathcal{F}_c$) in a communication-efficient way in all variants of the omissions model. We note that this representation depends heavily on the fact that the only faulty behavior a faulty processor may exhibit involves the loss of

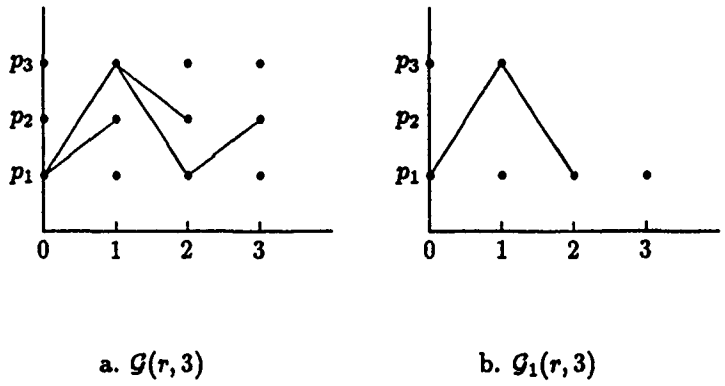


Figure 3.1: Communication graphs.

messages. The technique does not work in the Byzantine models where processors may send incorrect messages in addition to losing messages. Results of [Coa86, Mic88, Mic89] show how the size of messages in such models may be reduced.

Given a run r of $\mathcal{F}$, the *communication graph* of r (see Figure 3.1) represents the messages delivered in r. It is a layered graph (with one layer corresponding to every natural number, representing time on the global clock) in which each processor is represented by one node in every layer. We denote the node representing processor p at time ℓ by $\langle p, \ell \rangle$. Edges connect nodes in adjacent layers, with an edge between $\langle p, k-1 \rangle$ and $\langle q, k \rangle$ iff a message from p is delivered to q during round k. The *labeled communication graph* is obtained by labeling the layer 0 nodes of the communication graph with processors' names and initial states, and by labeling the layer k nodes (for $k > 0$) with the requests the processors receive from external clients during round k. We note in passing that, since r is a run of the full-information protocol $\mathcal{F}$, its labeled communication graph uniquely determines its operating environment. For every point (r, ℓ), we denote by $\mathcal{G}(r, \ell)$ the first $\ell + 1$ layers of the labeled communication graph of r, representing the first ℓ rounds of r. For example, illustrated in Figure 3.1(a) is a graph $\mathcal{G}(r, 3)$ depicting the first 3 rounds of a run r. We say that $\mathcal{G}(r, \ell)$ has *length* ℓ.

Informally, at every point (r, ℓ) of a run of $\mathcal{F}$, a processor p_i's local state corresponds to a certain subgraph $\mathcal{G}_i(r, \ell)$ of $\mathcal{G}(r, \ell)$. For example, the subgraph $\mathcal{G}_1(r, 3)$ of $\mathcal{G}(r, 3)$ is illustrated in Figure 3.1(b). We define the subgraph $\mathcal{G}_i(r, \ell)$ of $\mathcal{G}(r, \ell)$ inductively as follows. For $\ell = 0$ the subgraph $\mathcal{G}_i(r, 0)$ consists of the labeled node $\langle p_i, 0\rangle$. For $\ell > 0$ the subgraph $\mathcal{G}_i(r, \ell)$ consists of the labeled node $\langle p_i, \ell\rangle$, the subgraph $\mathcal{G}_i(r, \ell - 1)$, the edges from layer $\ell - 1$ nodes to $\langle p_i, \ell\rangle$, and the subgraphs $\mathcal{G}_j(r, \ell - 1)$ for every layer $\ell - 1$ node $\langle p_j, \ell - 1\rangle$ adjacent to $\langle p_i, \ell\rangle$. Given a set S of processors, it is convenient to denote by $\mathcal{G}_S(r, \ell)$ the union of the graphs $\mathcal{G}_i(r, \ell)$ for every $p_i \in S$. We remark that $\mathcal{G}_S(r, \ell)$ uniquely determines $\mathcal{G}_i(r, \ell)$ for every $p_i \in S$. The next lemma states that a processor's state of the labeled communication graph uniquely determines its view of the run.

Lemma 3.5: Let r and r' be runs of a full-information protocol $\mathcal{F}$. For every processor p_i and time ℓ, $r_i(\ell) = r'_i(\ell)$ iff $\mathcal{G}_i(r, \ell) = \mathcal{G}_i(r', \ell)$.

Proof: We proceed by induction on ℓ. The case of $\ell = 0$ is immediate. Suppose $\ell > 0$ and the inductive hypothesis holds for $\ell - 1$.

Suppose p_i has the same local state at time ℓ in both r and r'. This implies, in particular, that p_i has the same local state at time $\ell - 1$ in r and r', and from the inductive hypothesis it follows that $\mathcal{G}_i(r, \ell - 1) = \mathcal{G}_i(r', \ell - 1)$. In addition, this implies that p_i must receive the same input during round ℓ in r and r', and hence $\langle p_i, \ell\rangle$ is labeled with the same input in $\mathcal{G}_i(r, \ell)$ and $\mathcal{G}_i(r', \ell)$. If p_i does not receive a message from a processor p_j during round ℓ in r and r', then there is no edge from $\langle p_j, \ell - 1\rangle$ to $\langle p_i, \ell\rangle$ in either $\mathcal{G}_i(r, \ell)$ or $\mathcal{G}_i(r', \ell)$. If p_i *does* receive a message from a processor p_j during round ℓ in r and r', then it receives the same message in both runs, and p_j must have the same local state at time $\ell - 1$ in both runs. Hence, there is an edge from $\langle p_j, \ell - 1\rangle$ to $\langle p_i, \ell\rangle$ in both $\mathcal{G}_i(r, \ell)$ and $\mathcal{G}_i(r', \ell)$, and by the inductive hypothesis we have that $\mathcal{G}_j(r, \ell - 1) = \mathcal{G}_j(r', \ell - 1)$. Thus, $\mathcal{G}_i(r, \ell) = \mathcal{G}_i(r', \ell)$.

Conversely, suppose $\mathcal{G}_i(r, \ell) = \mathcal{G}_i(r', \ell)$. It follows that $\mathcal{G}_i(r, \ell - 1) = \mathcal{G}_i(r', \ell - 1)$, and by the inductive hypothesis p_i has the same local state at time $\ell - 1$ in r and r'. The node $\langle p_i, \ell\rangle$ must be labeled with the same input in $\mathcal{G}_i(r, \ell)$ and $\mathcal{G}_i(r', \ell)$, so p_i receives the same input during round ℓ in r and r'. The edges from layer $\ell - 1$ nodes to $\langle p_i, \ell\rangle$ are the same in $\mathcal{G}_i(r, \ell)$ and $\mathcal{G}_i(r', \ell)$, so p_i receives messages from the same processors during round ℓ in r and r'. Again, $\mathcal{G}_j(r, \ell - 1) = \mathcal{G}_j(r', \ell - 1)$ for every node $\langle p_j, \ell - 1\rangle$ adjacent to $\langle p_i, \ell\rangle$,

and by the inductive hypothesis p_j has the same local state at time $\ell - 1$ in r and r'. Since $\mathcal{F}$ requires that every processor send its entire local state in every round, p_i receives the same messages during round ℓ in r and r'. It follows that p_i has the same local state at time ℓ in both r and r'. □

Lemma 3.5 implies that a processor's local state and its view of the corresponding labeled communication graph convey the same information: Given either the graph $\mathcal{G}_i(r, \ell)$ or the local state $r_i(\ell)$, reconstructing the other is straightforward. Therefore, an equivalent implementation of a full-information protocol is one in which the processors send the labeled communication graphs corresponding to their local states instead of sending their entire local states. From now on, we will use the term *full-information protocol* to refer to this equivalent form. It is easy to see that the size of $\mathcal{G}_i(r, \ell)$ is polynomial in the number of processors n, the global time ℓ, and the size of the requests received from external clients. It follows that messages required by a full-information protocol are of polynomial size.[4] Furthermore, given the labeled communication graphs corresponding to the local states at time $\ell - 1$ of the processors that send messages to a given processor p_i during round ℓ, it is easy to construct the labeled communication graph corresponding to p_i's local state at time ℓ. Thus, the use of such compact representations of a processor's state is computationally efficient as well as communication efficient. Finally, recall that we have formally defined a test for common knowledge to be a function of processor states (as well as n and t). In light of the preceding discussion, there is no loss of generality in assuming that such a test is a function of communication graphs corresponding to processor states. We now turn to the problem of implementing such tests.

3.5 Testing for Common Knowledge

The previous section established the claim that tests for common knowledge provide a very powerful programming technique: The design of optimal protocols for simultaneous choice problems reduces to implementing tests for common knowledge of certain facts. In this section we investigate the problem of implementing tests for common knowledge in the different variants of

[4] In the Byzantine failure models, however, in which processors are allowed to lie (or maliciously deviate from the protocol), we know of no such compact representations. See [Coa86] for a trade-off between message size and running time possible in such models.

the omissions model. With such tests, we will be able to construct optimal protocols for simultaneous choice problems in these models. As we will see, properties of the different variants of the omissions model cause dramatic differences in the complexity of testing for common knowledge. In addition, the optimal protocols we construct will have interesting properties that vary according to the failure model. We will discuss these properties as we consider each variant later in this section.

Recall that a protocol is a function that, given the number of processors n, the bound t on the number of faulty processors, and a processor's state, yields a list of the actions the processor should perform, as well as the messages it should send in the next round. (Thus, the protocols we are interested in are uniform in n and t.) Since the protocols we will be concerned with are full-information protocols, processors' states will be efficiently representable by labeled communication graphs. We will soon restrict our attention to simultaneous choice problems in which the external requests are of constant size (or, equivalently, to problems involving only a constant number of possible requests from external clients). This restriction implies that processors' states at time ℓ will be of size polynomial in n and ℓ. A protocol will therefore determine the messages and actions required at time ℓ based on input of size polynomial in n and ℓ. Consequently, we will measure the complexity of computations performed by protocols at time ℓ in systems of n processors as a function of n and ℓ: By polynomial time, polynomial space, etc., we will mean polynomial in n and ℓ.

The definition of simultaneous choice problems presented in Section 3.3 is very general, so general, in fact, that it is possible to define simultaneous choice problems with a variety of anomalous properties. For example, it is possible to define a simultaneous choice problem in which $pro(a)$ is the fact φ = "the first round in which p receives an external request is a round whose number is the index of a halting Turing machine" (in some a priori well-defined enumeration of Turing machines), and $con(a)$ is $\neg\varphi$. Clearly, since it is undecidable whether φ holds even given the local state of p after it receives its first request, it will also be undecidable which of $C_N\varphi$ or $C_N\neg\varphi$ holds when processor p's local state becomes common knowledge. It follows that this simultaneous choice problem cannot be effectively implemented by a computable protocol. Similarly, one can construct simultaneous choice problems in which evaluation of the conditions is intractable, rather than undecidable as in the above example. It is also possible to introduce

anomalies by defining the sets Γ_i of external inputs in strange ways. Since we are not interested in problems involving such inherent anomalies, we will avoid them by making restrictions on the relevant facts and the inputs arising in the simultaneous choice problems we will consider in the sequel.

We first define the class of *practical* facts, which will be used to restrict the conditions that specify a simultaneous choice problem. Roughly speaking, one essential property of a practical fact φ is that it is easy to determine from a processor's state whether a run satisfies φ. More formally, we denote by "$\mathcal{G}_S(r,\ell)$" the property of being a run r' such that $\mathcal{G}_S(r,\ell) = \mathcal{G}_S(r',\ell)$. Consequently, if $\mathcal{G}_S(r,\ell) \supset \varphi$ is valid in a system, then every run r' of the system satisfying $\mathcal{G}_S(r,\ell) = \mathcal{G}_S(r',\ell)$ must also satisfy φ. In this case, we say that $\mathcal{G}_S(r,\ell)$ *determines* φ. Notice, for example, that no finite labeled communication graph $\mathcal{G}_S(r,\ell)$ can determine that a run is failure-free (since the run is infinite, and a failure can always happen outside the finite scope depicted by the graph). With this notion in mind, a fact φ is said to be *practical* within a class of systems $\{\Sigma(n,t) : n \geq t+2\}$ if the following conditions hold: (i) φ is a fact about the input and the existence of failures, and (ii) there is a polynomial-time algorithm to determine, given n, t, and a graph $\mathcal{G}_S(r,\ell)$ of a point of $\Sigma(n,t)$, whether $\mathcal{G}_S(r,\ell) \supset \varphi$ is valid in $\Sigma(n,t)$. The first condition is justified by the fact that we will be testing for common knowledge of the conditions *enabled*(a_i) arising from natural simultaneous choice problems, and such conditions are typically conditions on the input and existence of failures. The second condition ensures that it is easy to test whether a labeled communication graph determines that the fact holds. (We make this restriction since it would clearly be unreasonable to expect the processors to be able to efficiently identify and act based on facts that are intractable to compute from the labeled communication graph.)

We now consider a natural restriction on the sets Γ_i of possible inputs. A class of systems is said to be *practical* if there are two fixed finite sets S and M of initial states and external requests, respectively, such that each Γ_i in all systems of the class is the set of complete input histories whose initial state is in S, and in which the input received in every round is a subset of M. This condition ensures that the input sets are of a simple form. In particular, it implies that all Γ_i's are identical, and that the input received by a processor during any given round is of constant size.

Having defined the notions of practical facts and practical classes of systems, we say that a simultaneous choice $\mathcal{C}$ is *practical* if (i) the class of systems

determined by a full-information protocol and $\mathcal{C}$ is practical, and (ii) each condition *enabled*(a_i) is practical within this class of systems. Essentially all natural simultaneous choice problems are practical. In particular, all simultaneous choice problems appearing in the literature are practical. Our analysis will hence be restricted to testing for common knowledge of practical facts and to designing and implementing optimal protocols for practical simultaneous choice problems. We remark, however, that our analysis will apply to a more general class of simultaneous choice problems, whose precise characterization is somewhat complicated.

In Section 3.4 we programmed protocols for simultaneous choice problems in a high-level language in which processors' actions depend on explicit tests for common knowledge. Recall that *test-for-*$C_{\mathcal{N}}$*enabled*(a_i) is a test nonfaulty processors can use to determine whether *enabled*(a_i) is common knowledge: Given the graph corresponding to the local state of a nonfaulty processor at (r, ℓ) as input, *test-for-*$C_{\mathcal{N}}$*enabled*(a_i) returns **true** iff $(r, \ell) \models C_{\mathcal{N}}$*enabled*$(a_i)$. Theorem 3.4 implies that given such a test for each condition *enabled*(a_i), the protocol $\mathcal{F}_c$ is an optimal protocol for $\mathcal{C}$. Until this point, however, we have sidestepped the issue of whether such tests actually exist. With the next lemma we see that, for practical simultaneous choice problems, such tests can be implemented in polynomial space.

Lemma 3.6: If $\mathcal{C}$ is a practical simultaneous choice problem, then for each action a_i the test *test-for-*$C_{\mathcal{N}}$*enabled*(a_i) can be implemented in polynomial space.

Proof: We must prove the existence of an algorithm *test-for-*$C_{\mathcal{N}}$*enabled*(a_i) determining in polynomial space whether *enabled*(a_i) is common knowledge at (r, ℓ), given as input n, t, and the graph $\mathcal{G}_j(r, \ell)$ corresponding to the local state of a nonfaulty processor p_j at (r, ℓ). We will actually exhibit a nondeterministic, polynomial-space algorithm A_i determining whether *enabled*(a_i) is *not* common knowledge at (r, ℓ). Since NPSPACE=PSPACE and PSPACE is closed under complementation (see [HU85]), the existence of the algorithm A_i implies the existence of an algorithm *test-for-*$C_{\mathcal{N}}$*enabled*(a_i).

Let $\{\Sigma(n, t) : n \geq t + 2\}$ be a class of systems determined by a full-information protocol and the problem $\mathcal{C}$. We claim that such an algorithm A_i need only guess a point (s, ℓ) with the property that $\mathcal{G}(s, \ell) \supset$ *enabled*(a_i) is *not* valid in $\Sigma(n, t)$, guess the path from (r, ℓ) to (s, ℓ) in the similarity

graph proving that $(r,\ell) \sim (s,\ell)$, and then verify that these two conditions hold. To see this, notice that since $\mathcal{G}(s,\ell) \supset$ *enabled*(a_i) is *not* valid in the system, there must be a point (s',ℓ) such that $\mathcal{G}(s,\ell) = \mathcal{G}(s',\ell)$ and $(s',\ell) \not\models$ *enabled*(a_i). Construct the run with the input of s' in which processors fail precisely as they do in s for the first ℓ rounds, and in which no processor fails after time ℓ. Let u be a run obtained by adding to this run a single failure after time ℓ iff there is a failure in s'. Since u and s' must satisfy the same facts about the input and existence of failures, $(s',\ell) \not\models$ *enabled*(a_i) implies $(u,\ell) \not\models$ *enabled*(a_i). Since at least one nonfaulty processor in s is nonfaulty in u, and also has the same local state at time ℓ since $\mathcal{G}(u,\ell) = \mathcal{G}(s,\ell)$, we have $(s,\ell) \sim (u,\ell)$. Therefore, $(r,\ell) \sim (u,\ell)$ and $(u,\ell) \not\models$ *enabled*(a_i), and it follows that $(r,\ell) \not\models C_{\mathcal{N}}$*enabled*$(a_i)$.

We now describe the algorithm A_i in greater detail. Notice that since C is practical, the input received by a processor in every round of a run of $\Sigma(n,t)$ is of constant size, and hence it is possible to construct the labeled communication graph of any point of $\Sigma(n,t)$ in polynomial space.

The algorithm A_i first guesses the point (s,ℓ) and writes it down in polynomial space. Since *enabled*(a_i) is a practical fact, A_i can show in polynomial time (and hence in polynomial space) that $\mathcal{G}(s,\ell) \supset$ *enabled*(a_i) is *not* valid in the system $\Sigma(n,t)$.

The algorithm then guesses the path from (r,ℓ) to (s,ℓ) in the similarity graph step by step, verifying each step in polynomial-space as it goes. The algorithm A_i begins by constructing the graph $\mathcal{G}(r',\ell)$ of a run r' by adding to the graph $\mathcal{G}_j(r,\ell)$ received as input all edges not recorded as missing in $\mathcal{G}_j(r,\ell)$. Notice that since p_j is nonfaulty in r, it is nonfaulty in r' as well, and hence $(r,\ell) \sim (r',\ell)$. The algorithm A_i then shows that $(r',\ell) \sim (s,\ell)$ (and hence that $(r,\ell) \sim (s,\ell)$) in polynomial space by constructing one by one the graph $\mathcal{G}(u_i,\ell)$ of each point (u_i,ℓ) in a path from (r',ℓ) to (s,ℓ) in the similarity graph. For each pair of points (u_{i-1},ℓ) and (u_i,ℓ), the algorithm shows that some nonfaulty processor p_k has the same local state at both points by choosing p_k, exhibiting for each point an assignment of faulty processors (consistent with their respective graphs) in which p_k is nonfaulty, and showing that p_k has the same local state at both points by verifying $\mathcal{G}_k(u_{i-1},\ell) = \mathcal{G}_k(u_i,\ell)$. □

It is important to realize that Lemma 3.6 holds in all variants of the omissions model: The failure model is a parameter of a simultaneous choice

problem, and we have made no assumptions restricting the failure model in this result. We note that the proof of Lemma 3.6 actually shows that testing for common knowledge of any practical fact can be done in polynomial space. In fact, the proof shows that such tests have effective implementations even when the algorithm determining whether $\mathcal{G}(r, \ell) \supset enabled(a_i)$ is valid does not run in polynomial time (although the problem must still be decidable). In this case, however, the test is guaranteed to run in polynomial space only if this computation can be performed using polynomial space. The most important consequence of Lemma 3.6, however, is that practical simultaneous choice problems have polynomial-space optimal protocols.

Theorem 3.7: If $\mathcal{C}$ is an implementable practical simultaneous choice problem, then there is a polynomial-space optimal protocol for $\mathcal{C}$.

With Theorem 3.7 we see that practical simultaneous choice problems do have effective optimal protocols. In general, however, connected components in the similarity graph may be of exponential size, and paths in such components may be of exponential length. It therefore follows that the polynomial-space protocol given by Theorem 3.7 requires the processors to perform exponential-time computations between consecutive rounds of communication. The resulting protocol is therefore clearly not a reasonable protocol to use in practice. A crucial question at this point is whether there are *efficient* optimal protocols for simultaneous choice problems. Recall that we have already seen that optimal protocols can be implemented in a way that makes efficient use of communication. The rest of the chapter is devoted to investigating ways of implementing tests for common knowledge in variants of the omissions model in a computationally-efficient manner, and therefore of implementing efficient, optimal protocols for simultaneous choice problems in these models.

3.5.1 The Omissions Model

Recall that in the omission model a faulty processor may fail only by failing to send some of the messages its protocol requires it to send. In this section we consider the problem of efficiently implementing tests for common knowledge in the omissions failure model. In particular, we develop a construction that crisply characterizes the connected component of a point in

the similarity graph. This construction determines a subgraph of the labeled communication graph with the property that two points are similar iff their respective subgraphs are identical. As stated in Theorem 2.3, the connected component of a point in the similarity graph completely determines what facts are common knowledge at that point. As a result, this construction enables us to devise efficient tests for common knowledge, and hence efficient protocols for simultaneous choice problems that are optimal in all runs.

Dwork and Moses address in [DM86] the problem of implementing tests for common knowledge in the *crash failure model.* In the crash failure model, processors fail by crashing; that is, faulty processors may successfully send messages to some processors during their failing round, but will not successfully send any messages in any later round. As a result, a faulty processor is "out of the game" after its failing round, and no longer contributes to the knowledge of the remaining processors. The analysis performed by Dwork and Moses focuses on the notion of a *clean round*, a round in which no processor failure is discovered. In runs of a full-information protocol, a clean round ensures that all nonfaulty processors receive the same set of messages. After such a round, all nonfaulty processors have an identical view of the part of the run that precedes the clean round. Dwork and Moses show that facts about the initial configuration become common knowledge exactly when it becomes common knowledge a clean round has occurred. Dwork and Moses complete their analysis by characterizing when this happens. In the omissions model, however, because a faulty processor need not remain silent, or crash, after first failing to send a message), a faulty processor may continue to contribute to the knowledge of the nonfaulty processors, even after its first failing round. The situation is therefore more complicated, and clean rounds no longer play the same role here as they do in the crash failure model. Furthermore, to the best of our understanding, there is no direct analog to the notion of a clean round in the omissions model. The approach used by Dwork and Moses in the crash failure model, therefore, does not seem to extend to this model. As a result, we are forced to take a different approach.[5]

[5]As mentioned in the introduction, since the technical details of the proofs in this section may make it difficult to obtain a high-level understanding of our approach, we encourage the reader to skip the proofs on the first reading.

The Basic Steps

We now give what will become the two basic steps of our test for common knowledge during runs of a full-information protocol in the omissions model. (Unless otherwise mentioned, all protocols referred to in this section will be full-information protocols.) Our approach to the problem of testing for common knowledge is motivated by a careful analysis of what facts do *not* become common knowledge. We begin with a technical result, similar to Lemma 15 of [DM86], saying that two points are similar if they differ only in the faulty behavior exhibited by a single processor in the last few rounds.

Throughout the remainder of this chapter it will be convenient to refer to runs differing only in some aspect of their operating environments. Given two runs r and r' of a protocol $\mathcal{F}$, we will say that r *differs from* r' *only in* a certain aspect of the operating environment if r is the result of executing $\mathcal{F}$ in an operating environment that differs from that of r' only in the said aspect. Notice that while their operating environments may be similar, the messages sent in the two runs may actually be quite different. We say that a processor is *silent* from time k if it fails to send every message in every round following time k.

Lemma 3.8: Let r and r' be runs differing only in the (faulty) behavior displayed by processor p after time k, and suppose no more than f processors fail in either r or r'. If $\ell - k \leq t + 1 - f$, then $(r, \ell) \sim (r', \ell)$.

Proof: If $k \geq \ell$ then $\mathcal{G}(r, \ell) = \mathcal{G}(r', \ell)$, and Lemma 3.5 implies that $(r, \ell) \sim (r', \ell)$. Therefore, assume $k < \ell$. We proceed by induction on $j = \ell - k$. Without loss of generality, we may assume that r and r' actually differ in the faulty behavior of p, and hence that p fails in one of these runs. Notice that since p already fails in one of these runs and yet no more than f processors fail in either run, it is clear that at most $f \leq t$ processors fail in any run differing from either run only in the faulty behavior of p.

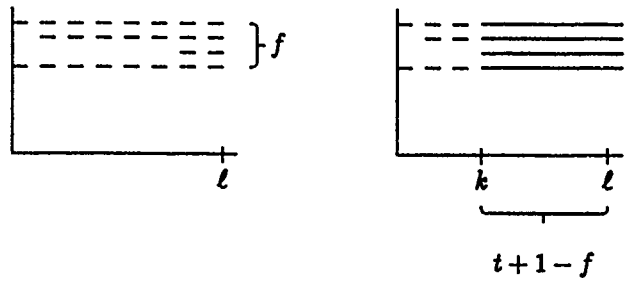
Suppose $j = 1$ (that is, $k = \ell - 1$). Since $t \leq n - 2$ and since r and r' differ only in the behavior of p, there are two processors q_1 and q_2 (other than p) that do not fail in either run. Let r_2 be the run differing from r only in that p sends to q_2 during round ℓ of r_2 iff it does so in r' (and notice that r_2 may actually be equal to r). Since q_1's local state at time ℓ is independent of whether p sends to q_2 during round ℓ, we have $(r, \ell) \sim (r_2, \ell)$. Since $\mathcal{G}(r_2, \ell)$ and $\mathcal{G}(r', \ell)$ differ only in the messages that p sends to processors other than q_2

in round ℓ, and q_2's local state at (r_2, ℓ) is independent of whether p sends to the remaining processors during round ℓ, we have $(r_2, \ell) \sim (r', \ell)$. Thus, by the transitivity of "$\sim$," we have $(r, \ell) \sim (r', \ell)$.

Suppose $j > 1$ (that is, $k < \ell - 1$) and the inductive hypothesis holds for $j - 1$. Let r_i be the run differing from r only in that for each processor q in $\{p_1, \ldots, p_i\}$ processor p sends to q during round $k + 1$ in r_i iff it does so in r'. Notice that $r = r_0$. We will show that $(r, \ell) \sim (r_i, \ell)$ for all $i \geq 0$. Since r_n differs from r' only in the faulty behavior of p after time $k + 1$, and since $\ell - (k + 1) = j - 1$, it will follow by the inductive hypothesis for $j - 1$ that $(r_n, \ell) \sim (r', \ell)$. Finally, by the transitivity of "$\sim$," we will have $(r, \ell) \sim (r', \ell)$ as desired.

We now proceed by induction on i to show that $(r, \ell) \sim (r_i, \ell)$ for all $i \geq 0$. The case of $i = 0$ is trivial. Suppose $i > 0$ and the inductive hypothesis holds for $i - 1$; that is, $(r, \ell) \sim (r_{i-1}, \ell)$. Notice r_{i-1} and r_i differ at most in whether p sends a message to p_i during round $k + 1$. Let s be the run differing from r_{i-1} in that p_i is silent from time $k + 1$ in s. Suppose no more than g processors fail in either r_{i-1} or s. Notice that $g \leq f + 1$. Therefore, since $1 < \ell - k \leq t + 1 - f$ we have $f < t$ and $g \leq t$, so at most t processors fail in s. Furthermore, $\ell - (k + 1) \leq t + 1 - (f + 1) \leq t + 1 - g$. Since, in addition, r_{i-1} and s differ only in the faulty behavior of p_i after time $k + 1$, the inductive hypothesis for $j - 1$ implies $(r_{i-1}, \ell) \sim (s, \ell)$. Now, since p_i is silent from time $k + 1$ in s, the local state of a nonfaulty processor at (s, ℓ) is independent of whether p sends to p_i during round $k + 1$, so $(s, \ell) \sim (s', \ell)$ where s' differs from s in that p sends to p_i during round $k + 1$ in s' iff it does so in r_i. Again, the inductive hypothesis for $j - 1$ implies that $(s', \ell) \sim (r_i, \ell)$. By the transitivity of "$\sim$," it follows that $(r, \ell) \sim (r_i, \ell)$. □

While Lemma 3.8 is a technical lemma in the context of this work, it has a number of interesting consequences in its own right. In particular, the $(t+1)$-round lower bound on the number of rounds required for simultaneous Byzantine agreement is an immediate corollary of this lemma. The resulting proof of this lower bound is perhaps the simplest to appear in the literature (see [DM86] for details). More important for our purposes, however, is the fact that two corollaries of Lemma 3.8 enable us to characterize the connected components of the similarity graph. Consider the runs r_1 and r_2 of Figure 3.2, where we indicate only faulty behavior: solid lines indicate silence, and dashed lines indicate sporadic faulty behavior. Notice that f



The run r_1. The run r_2.

Figure 3.2: Runs illustrating Lemma 3.9.

processors fail in r_1. In the following lemma we show that $(r_1, \ell) \sim (r_2, \ell)$ where r_2 differs from r_1 only in that processors failing in r_1 are silent in r_2 from time k, where $k = \ell - (t + 1 - f)$. This is the first basic step of our test for common knowledge.

Lemma 3.9: Let r_1 be a run in which f processors fail. Let r_2 be the run differing from r_1 only in that processors failing in r_1 are silent from time k in r_2, where $k = \ell - (t + 1 - f)$. Then $(r_1, \ell) \sim (r_2, \ell)$.

Proof: Let $q_1, \ldots, q_f$ be the faulty processors in r_1. Let s_i be the run differing from r_1 in that processors $q_1, \ldots, q_i$ are silent from time k in s_i. Notice that $r_1 = s_0$ and $r_2 = s_f$. We proceed by induction on i to show that $(r_1, \ell) \sim (s_i, \ell)$ for all i. The case of $i = 0$ is trivial. Suppose $i > 0$ and the inductive hypothesis holds for $i - 1$; that is, $(r_1, \ell) \sim (s_{i-1}, \ell)$. Since s_{i-1} and s_i differ at most in the faulty behavior of q_i after time k, it follows by Lemma 3.8 that $(s_{i-1}, \ell) \sim (s_i, \ell)$. By the transitivity of "$\sim$," we have $(r_1, \ell) \sim (s_i, \ell)$. □

One interesting consequence of this result, for example, is that the states at time k of processors failing in r_1 are not common knowledge at time ℓ. To see this, let F be the set of processors failing in r_1, and suppose it is common knowledge at (r_1, ℓ) that "the joint view of F at time k is equal

to $r_{1F}(k)$." This means that this statement is true at all points in (r_1, ℓ)'s connected component. But let r_1' and r_2' be runs differing from r_1 and r_2 only in that some already-faulty processor $p \in F$ fails to send to another already-faulty processor $q \in F$ during round k. Notice that the joint view of F at time k in r_1' is *not* equal to $r_{1F}(k)$. Yet according to our lemma, $(r_1, \ell) \sim (r_2, \ell)$ and $(r_1', \ell) \sim (r_2', \ell)$; and since the processors in F are silent from time k, the points (r_2, ℓ) and (r_2', ℓ) are indistinguishable to all nonfaulty processors; and so $(r_2, \ell) \sim (r_2', \ell)$, which implies $(r_1, \ell) \sim (r_1', \ell)$, and hence (r_1, ℓ) and (r_1', ℓ) are in the same connected component! Consequently, the time k views of processors in the set F cannot be common knowledge at (r_1, ℓ). Interestingly, our next result will show that even the identity of F itself (the identity of the faulty processors) may not be common knowledge at (r_1, ℓ).

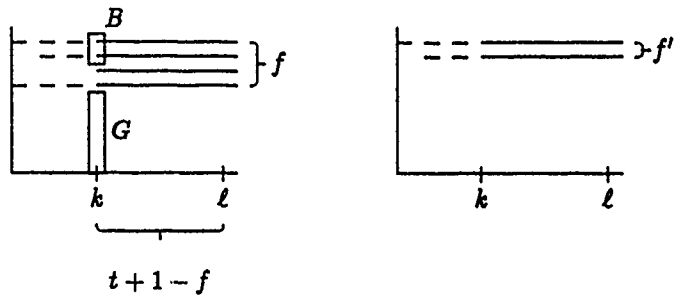
Before discussing the second lemma, however, we make an important definition. Given a point (r, k) and a set of processors G, let

$$B(G, r, k) \stackrel{\text{def}}{=} \{p : (r, k) \models I_G(\text{"}p \textit{ is faulty}\text{"})\}.$$

By this definition, $B(G, r, k)$ is the set of processors implicitly known by G at (r, k) to be faulty. An important property of the omissions failure model is that processors fail *only* by failing to send messages. It follows that G implicitly knows at (r, k) that a processor p is faulty iff G implicitly knows at (r, k) of some processor q not receiving a message from p at time k or earlier; that is, $\mathcal{G}_G(r, k)$ contains no edge from $\langle p, \ell - 1\rangle$ to $\langle q, \ell\rangle$ for some node $\langle q, \ell\rangle$ of $\mathcal{G}_G(r, k)$. It is therefore simple and straightforward to compute $B(G, r, k)$ given $\mathcal{G}_G(r, k)$.

The essence of the second lemma is captured by the runs r_2 and r_3 of Figure 3.3. In the run r_2, the f faulty processors are silent from time $k = \ell - (t+1-f)$. The set G is the set of nonfaulty processors and $B = B(G, r_2, k)$. The run r_3 differs from r_2 only in that processors in $P - B$ do not fail in r_3. The following lemma states that $(r_2, \ell) \sim (r_3, \ell)$. This implies, for instance, that the failure of processors in $P - B$ cannot be common knowledge at (r_2, ℓ) since they do not fail in r_3. Formally, the second basic step of our test for common knowledge can be stated as follows (see Figure 3.3):

Lemma 3.10: Let r_2 be a run in which the f faulty processors are silent from time $k = \ell - (t + 1 - f)$. Let G be the set of nonfaulty processors in



The run r_2. The run r_3.

Figure 3.3: Runs illustrating Lemma 3.10.

r_2, and let $B = B(G, r_2, k)$. Let r_3 be the run differing from r_2 only in that processors in $P - B$ do not fail. Then $(r_2, \ell) \sim (r_3, \ell)$.

Proof: If a processor p in $P - B$ fails to a processor q during some round $j \leq k$ of r_2 (in which case it must be that $p \in P - B - G$), then the node $\langle q, j \rangle$ must not be a node of $\mathcal{G}_G(r_2, k)$ or the failure of p would be implicitly known by G at time k and p would be in B, a contradiction. Thus, $\mathcal{G}_G(r_2, k)$ is independent of whether $\mathcal{G}(r_2, k)$ contains an edge from p to q during round j. Let r_2' be a run differing from r_2 only in that no processor in $P - B$ fails *before* time k in r_2'. By the previous discussion, $\mathcal{G}_G(r_2, k) = \mathcal{G}_G(r_2', k)$. In both r_2 and r_2' every processor in G successfully sends every message after time k and every processor in $P - G$ is silent from time k. Since, in addition, every processor in G receives the same input after time k in r_2 and r_2', we have $\mathcal{G}_G(r_2, \ell) = \mathcal{G}_G(r_2', \ell)$. Given that G is the set of nonfaulty processors in r_2, each of which is also nonfaulty in r_2', it follows by Lemma 3.5 that $(r_2, \ell) \sim (r_2', \ell)$. Since the runs r_2' and r_3 differ only in the faulty behavior of processors in $P - B$ after time k, by repeated application of Lemma 3.8 it follows that $(r_2', \ell) \sim (r_3, \ell)$. Hence, $(r_2, \ell) \sim (r_3, \ell)$. □

Characterizing the Similarity Graph

Let us now consider how these two basic steps, Lemmas 3.9 and 3.10, can be used to characterize the connected components of the similarity graph, and hence what facts are common knowledge at a given point. Going back to Figures 3.2 and 3.3, notice that if $f' < f$ (which implies, referring to Figure 3.3, that not all f processors failing in r_1 are implicitly known at time $k = \ell - (t + 1 - f)$ to be faulty), then by setting $r_1' = r_3$ we can apply Lemmas 3.9 and 3.10 again (this time starting from r_1' instead of r_1). Iterating this process, we reach a run $\hat{r}$ satisfying $(r_1, \ell) \sim (\hat{r}, \ell)$, where the $\hat{f}$ processors failing in $\hat{r}$ are silent from time $\hat{k} = \ell - (t + 1 - \hat{f})$, and where all faulty processors are implicitly known to be faulty by the nonfaulty processors at $(\hat{r}, \hat{k})$. This run $\hat{r}$ is a fixpoint of this iterative process; setting $\hat{r}_1 = \hat{r}$, the runs $\hat{r}_2$ and $\hat{r}_3$ constructed in Lemmas 3.9 and 3.10 are identical to $\hat{r}$. It is the joint view of the nonfaulty processors at $(\hat{r}, \hat{k})$, we will show, that characterizes the connected component of (r_1, ℓ) in the similarity graph, and hence what facts are common knowledge at (r_1, ℓ). To enable ourselves to turn this characterization into a test for common knowledge individual processors can compute locally, we now define a local version of this iterative process, illustrated in Figure 3.4, that individual processors can use to construct locally this joint view.

Given a point (r, ℓ) and a processor p, this construction is defined as follows. Define $G_0 = \{p\}$ and $k_0 = \ell$, and define G_{i+1} and k_{i+1} inductively as follows. Denoting $B(G_i, r, k_i)$ by B_i, let

$$G_{i+1} = P - B_i$$

$$k_{i+1} = \ell - (t + 1 - |B_i|).$$

One should ask what happens to this construction when k_{i+1} becomes negative. Recall that when $k_{i+1} < 0$, the local state at time k_{i+1} of every processor in G_{i+1} is the distinguished *empty* local state. It follows that when $k_{i+1} < 0$, the set B_{i+1} must be empty. As a consequence, for all $j > i + 1$, we have that $G_j = P$, $k_j = \ell - (t + 1)$, and B_j is empty.

While we claim this is a construction each processor can perform locally, the set $B(G_i, r, k_i)$ is defined in terms of r, which individual processors cannot possibly know. We will soon show, however, that individual processors have enough information in their local state to compute $B(G_i, r, k_i)$ without

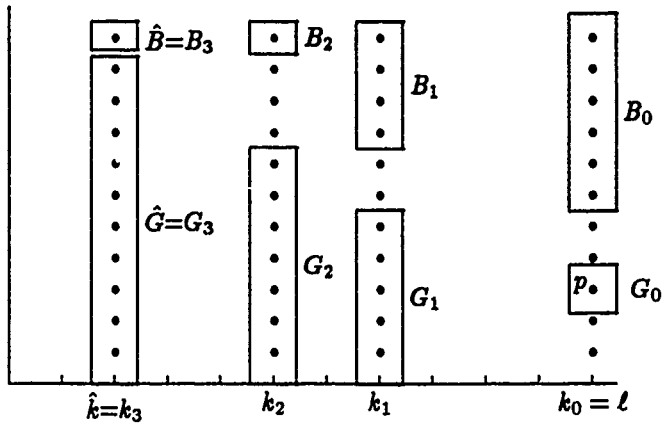


Figure 3.4: An example of the construction when $t = 9$.

knowing the precise identity of r, and hence can perform the steps of this construction locally.

The construction determines three (infinite) sequences $\{G_i\}$, $\{k_i\}$, and $\{B_i\}$. In the next few pages we will see that these sequences have limits $\hat{G}$, $\hat{k}$, and $\hat{B}$, and that these limits are independent of the processor with which the construction is begun. As a result, individual processors will be able to construct these values based solely on their local state. We will see that the joint view of $\hat{G}$ at time $\hat{k}$ completely characterizes the connected component of (r, ℓ) in the similarity graph, and hence what facts are common knowledge at (r, ℓ). This construction will therefore provide an efficient way of determining what facts are common knowledge at a given point.

Among other things, this construction captures a number of essential aspects of the information flow during the run up to time ℓ. In particular, one important property of this construction is the following:

Lemma 3.11: Every processor in G_{i+1} successfully sends to every processor in G_i in every round before time k_i.

Proof: Suppose some processor q of G_{i+1} fails to send to a processor q' of G_i during a round before time k_i. Then q's failure to q' is implicitly known by G_i at time k_i, so $q \in B_i$ and $q \notin G_{i+1}$, a contradiction. □

One consequence of Lemma 3.11 is that the local state of the processor p at time ℓ must contain the local state of every processor in G_i at time k_i for every $i \geq 0$. One property of the construction, therefore, is that the construction depends *only* on the local state of processor p at (r, ℓ), and hence that p is able to perform the construction locally. This property is essential in order to use this construction in a test for common knowledge that p can perform locally. A second essential property of the construction is that it converges within $t+1$ iterations, as we see with the following result.

Lemma 3.12: $\lim_{i \to \infty} G_i = G_{t+1}$ and $\lim_{i \to \infty} k_i = k_{t+1}$.

Proof: We will show that $B_{i+1} \subseteq B_i$ for all $i \geq 0$. Since B_0 contains at most t processors, it will then follow that there must be an $i \leq t$ for which $B_i = B_{i+1}$. From the definition of the construction, it is easy to see that we will have $B_i = B_{i+j}$ for all $j \geq 0$. In addition, we will have $G_{i+1} = G_{i+1+j}$ and $k_{i+1} = k_{i+1+j}$ for all $j \geq 0$, and we will be done. We proceed by induction on i. If $k_{i+1} < 0$, then B_{i+1} is empty and $B_{i+1} \subseteq B_i$, so let us assume $k_{i+1} \geq 0$. Suppose $i = 0$. By Lemma 3.11, every processor in G_1 must send to every processor in G_0 during round $k_1 + 1$. It follows that any failure implicitly known by G_1 at time k_1 must be implicitly known by G_0 at time k_0. Thus, $B_1 \subseteq B_0$. Suppose $i > 0$ and the inductive hypothesis holds for $i-1$; that is, $B_i \subseteq B_{i-1}$. If $B_i = B_{i-1}$, then $B_{i+1} = B_i$. If $B_i \subset B_{i-1}$, then $k_{i+1} < k_i$. By Lemma 3.11, G_{i+1} sends to G_i during round $k_{i+1} + 1$, so $B_{i+1} \subseteq B_i$. $\square$

We denote the results of the construction (the limits of the sequences $\{G_i\}$, $\{k_i\}$, and $\{B_i\}$) by $\hat{G}$, $\hat{k}$, and $\hat{B}$. We denote these values by $\hat{G}(p, r, \ell)$, $\hat{k}(p, r, \ell)$, and $\hat{B}(p, r, \ell)$ when the processor p and the point (r, ℓ) are not clear from context. We now show, however, that these values are independent of the processor p.

Lemma 3.13: $\hat{G}(p, r, \ell) = \hat{G}(q, r, \ell)$ and $\hat{k}(p, r, \ell) = \hat{k}(q, r, \ell)$ for all processors p and q.

Proof: We prove the claim by showing that $\hat{B}(p, r, \ell) = \hat{B}(q, r, \ell)$. Given that B_i uniquely determines G_{i+1} and k_{i+1}, this will imply the desired result. It suffices to show that $\hat{B}(p, r, \ell) \subseteq \hat{B}(q, r, \ell)$, since the other direction will follow by symmetry. Denote the intermediate results of the construction from the point (r, ℓ) starting with the processor p by G_i, k_i, and B_i, and the

final results by $\hat{G}$, $\hat{k}$, and $\hat{B}$. Similarly, denote the intermediate results of the construction starting with q by G'_i, k'_i, and B'_i, and the final results by $\hat{G}'$, $\hat{k}'$, and $\hat{B}'$. We now show that $\hat{B} \subseteq \hat{B}'$. If $\hat{k} < 0$, then $\hat{B}$ is empty and $\hat{B} \subseteq \hat{B}'$, so assume $\hat{k} \geq 0$. We consider two cases. First, suppose $\hat{k} = \ell - 1$. In this case, $\hat{B}$ must contain t faulty processors since $\hat{k} = \ell - (t + 1 - |\hat{B}|)$. It follows that every processor in $\hat{G}$ must be nonfaulty in r and hence must send to G'_0 during round $\hat{k} + 1$, so $\hat{B} \subseteq B'_0$. Since, in addition, $|B'_0| \leq t$ and $|\hat{B}| = t$, we have $\hat{B} = B'_0$. It follows from the construction that $\hat{B} = B'_i$ for every $i \geq 0$, and hence that $\hat{B} = \hat{B}'$.

Now, suppose $\hat{k} < \ell - 1$. Let q' be an (arbitrary) nonfaulty processor in r. We claim that every processor g in $\hat{G}$ must send its local state to q' during round $\hat{k} + 1$. Suppose some processor g in $\hat{G}$ does not. Let j be the least integer such that $\hat{G} = G_j$. If $j = 1$, then q' must send to G_0 during round $\hat{k} + 2$. If $j > 1$, then q' must actually be a member of G_{j-1} since G_{j-1} must contain all of the nonfaulty processors. In either case, the failure of g to q' during round $\hat{k} + 1$ must be implicitly known by G_{j-1} at time k_{j-1}, so $g \in B_{j-1}$. Since $\hat{G} = G_j = P - B_{j-1}$, we have $g \notin \hat{G}$, a contradiction. Thus, every processor in $\hat{G}$ must send to q' during round $\hat{k} + 1$.

We now proceed by induction on i to show that $\hat{B} \subseteq B'_i$ for all $i \geq 0$. Suppose $i = 0$. Every processor in $\hat{G}$ must send to the nonfaulty processor q' during round $\hat{k} + 1$, and q' must send to G'_0 during round $\hat{k} + 2$, so $\hat{B} \subseteq B'_0$. Suppose $i > 0$ and the inductive hypothesis holds for $i - 1$; that is, $\hat{B} \subseteq B'_{i-1}$. If $\hat{B} = B'_{i-1}$, then $\hat{B} = B'_i$. If $\hat{B} \subset B'_{i-1}$, then $\hat{k} < k'_i$ since $\hat{k} = \ell - (t + 1 - |\hat{B}|)$ and $k'_i = \ell - (t + 1 - |B'_{i-1}|)$. Every processor in $\hat{G}$ must send to the nonfaulty processor q' during round $\hat{k} + 1$, and q' must be contained in G'_i, so $\hat{B} \subseteq B'_i$. It follows that $\hat{B} \subseteq B'_i$ for all $i \geq 0$, and hence $\hat{B} \subseteq \hat{B}'$. □

As a result of Lemma 3.13, we see that $\hat{G}$, $\hat{k}$, and $\hat{B}$ depend only on the point (r, ℓ), and not the processor with which the construction begins. Thus, a third property of this construction is that *every* processor (and not just the nonfaulty processors) is able to compute locally the values of $\hat{G}$, $\hat{k}$, and $\hat{B}$. The ability of the nonfaulty processors to compute these values locally will be essential to designing a locally-computable test for common knowledge. We will denote these values by $\hat{G}(r, \ell)$, $\hat{k}(r, \ell)$, and $\hat{B}(r, \ell)$ when (r, ℓ) is not clear from context. From the definition of the construction it is clear that the driving force behind the construction is the identity of the sets B_i. Notice that these sets are uniquely determined by the failure pattern, and do not

depend on the run's input. Taking into account the input of a run, we are now in a position to show how the construction characterizes the connected components in the similarity graph. Denoting $\hat{G}(r,\ell)$ by $\hat{G}$ and $\hat{k}(r,\ell)$ by $\hat{k}$, we define

$$\hat{V}(r,\ell) \stackrel{\text{def}}{=} r_{\hat{G}}(\hat{k}).$$

This definition says that $\hat{V}(r,\ell)$ is the joint view of the processors in $\hat{G}(r,\ell)$ at time $\hat{k}(r,\ell)$. Our next lemma states that $\hat{V}$ is the same at similar points, which implies that the joint view $\hat{V}(r,\ell)$ is common knowledge at (r,ℓ).

Lemma 3.14: If $(r,\ell) \sim (r',\ell)$ then $\hat{V}(r,\ell) = \hat{V}(r',\ell)$.

Proof: We proceed by induction on the distance d between the points (r,ℓ) and (r',ℓ). The case of $d = 0$ is trivial. Suppose that $d > 0$ and the inductive hypothesis holds for $d-1$. Since the distance between (r',ℓ) and (r,ℓ) is d, there must be a point (s,ℓ) whose distance from (r,ℓ) is $d-1$, and whose distance from (r',ℓ) is 1. The inductive hypothesis implies that $\hat{V}(r,\ell) = \hat{V}(s,\ell)$, and we have $v(p,s,\ell) = v(p,r',\ell)$ for some processor p. As a consequence of Lemmas 3.11 and 3.13, the values of $\hat{V}(s,\ell)$ and $\hat{V}(r',\ell)$ depend only on the local state of p at (s,ℓ) and (r',ℓ), respectively. Since p has the same local state at (s,ℓ) and at (r',ℓ), we have $\hat{V}(s,\ell) = \hat{V}(r',\ell)$. Since $\hat{V}(r,\ell) = \hat{V}(s,\ell)$, it follows that $\hat{V}(r,\ell) = \hat{V}(r',\ell)$. □

One consequence of Lemma 3.14, together with Lemma 3.5 and the definition of $\hat{V}$ above, is that if $(r,\ell) \sim (r',\ell)$, then $\mathcal{G}_{\hat{G}}(r,\hat{k}) = \mathcal{G}_{\hat{G}}(r',\hat{k})$. We will find this a useful fact when proving the converse of Lemma 3.14; that is, that all points with the same $\hat{V}$ are similar, and hence that $\hat{V}$ completely characterizes the connected components of the similarity graph. Before we do so, however, let us formalize the intuition that led us to the construction in the first place (the use of the two basic steps in the construction given by Lemmas 3.9 and 3.10).

Lemma 3.15: Let r be a run, and let $\hat{G}$, $\hat{k}$, and $\hat{B}$ be the results of the construction from (r,ℓ). Let r' be the run differing from r only in that processors in $\hat{G}$ do not fail in r' and processors in $\hat{B}$ are silent from time $\hat{k}$ in r'. Then $(r,\ell) \sim (r',\ell)$.

Proof: Let G_i, k_i, and B_i be the intermediate results of the construction from (r,ℓ) starting with the nonfaulty processor p_j. For $i \geq 0$, define r_i to

be the run differing from the run r only in that processors in B_i are silent from time k_i in r_i and the remaining processors do not fail in r_i. Notice that $r' = r_i$ for sufficiently large i. We proceed by induction on i to show that $(r,\ell) \sim (r_i,\ell)$ for all $i \geq 0$. Suppose $i = 0$. Since the subgraph $\mathcal{G}_j(r,\ell)$ must be independent of whether the graph $\mathcal{G}(r,\ell)$ is missing an edge from a processor in $P - B_0$ to a processor other than p_j, we have $\mathcal{G}_j(r,\ell) = \mathcal{G}_j(r_0,k_0)$. Since processor p_j is nonfaulty, it follows that $(r,\ell) \sim (r_0,\ell)$. Suppose $i > 0$ and the inductive hypothesis holds for $i-1$; that is, $(r,\ell) \sim (r_{i-1},\ell)$. Lemma 3.9 implies $(r_{i-1},\ell) \sim (r'_{i-1},\ell)$ where r'_{i-1} differs from r_{i-1} in that processors in B_{i-1} (the processors failing in r_{i-1}) are silent from time k_i in r'_{i-1}. Lemma 3.10 implies $(r'_{i-1},\ell) \sim (r_i,\ell)$. Thus, $(r,\ell) \sim (r_i,\ell)$. □

Finally, we have the following:

Lemma 3.16: If $\hat{V}(r,\ell) = \hat{V}(r',\ell)$ then $(r,\ell) \sim (r',\ell)$.

Proof: The fact $\hat{V}(r,\ell) = \hat{V}(r',\ell)$ implies $\hat{G}(r,\ell) = \hat{G}(r',\ell)$, $\hat{k}(r,\ell) = \hat{k}(r',\ell)$, and $\hat{B}(r,\ell) = \hat{B}(r',\ell)$. We therefore denote these values by $\hat{G}$, $\hat{k}$, and $\hat{B}$. Let s be a run that differs from r in that processors in $\hat{G}$ do not fail in s, and processors in $\hat{B}$ are silent from time $\hat{k}$ in s. Let s' be an analogous run with respect to r'. Lemma 3.15 implies that $(r,\ell) \sim (s,\ell)$ and $(r',\ell) \sim (s',\ell)$. In order to show that $(r,\ell) \sim (r',\ell)$, it is enough to show that $(s,\ell) \sim (s',\ell)$. Suppose $\hat{G} = \{q_1,\ldots,q_m\}$, and let s_i be the run differing from s in that $q_1,\ldots,q_i$ receive the same input after time $\hat{k}$ in s_i as they do in s'. We proceed by induction on i to show that $(s,\ell) \sim (s_i,\ell)$ for all $i \geq 0$. Since $s = s_0$, the case of $i = 0$ is trivial. Suppose $i > 0$ and the inductive hypothesis holds for $i-1$; that is, $(s,\ell) \sim (s_{i-1},\ell)$. Let u_{i-1} and u_i be runs differing from s_{i-1} and s_i, respectively, only in that q_i is silent from time $\hat{k}$ in u_{i-1} and u_i. Lemma 3.8 implies $(s_{i-1},\ell) \sim (u_{i-1},\ell)$ and $(s_i,\ell) \sim (u_i,\ell)$. In addition, since u_{i-1} and u_i differ only in the input received by q_i after time $\hat{k}$, and since q_i is silent from time $\hat{k}$ in both runs, we have $(u_{i-1},\ell) \sim (u_i,\ell)$. Thus, $(s,\ell) \sim (s_i,\ell)$ for all $i \geq 0$. In particular, $(s,\ell) \sim (s_m,\ell)$. In order to complete the proof, it now suffices to show that $(s_m,\ell) \sim (s',\ell)$. Since $\mathcal{G}_{\hat{G}}(r,\hat{k}) = \mathcal{G}_{\hat{G}}(r',\hat{k})$, $(r,\ell) \sim (s,\ell)$, and $(r',\ell) \sim (s',\ell)$, Lemma 3.14 implies that $\mathcal{G}_{\hat{G}}(s,\hat{k}) = \mathcal{G}_{\hat{G}}(s',\hat{k})$. Notice that $\mathcal{G}_{\hat{G}}(s_m,\hat{k}) = \mathcal{G}_{\hat{G}}(s,\hat{k}) = \mathcal{G}_{\hat{G}}(s',\hat{k})$. Notice, in addition, that processors in $\hat{G}$ do not fail in either s_m or s', and that the remaining processors (in $\hat{B}$) are silent from time $\hat{k}$ in both runs. Finally, notice that processors in $\hat{G}$ receive the same input after time $\hat{k}$ in both runs.

It follows that $\mathcal{G}_{\hat{G}}(s_m, \ell) = \mathcal{G}_{\hat{G}}(s', \ell)$, and hence that $(s_m, \ell) \sim (s', \ell)$. Thus, $(s, \ell) \sim (s', \ell)$, as desired. □

Combining Lemmas 3.14 and 3.16 we see that $(r, \ell) \sim (r', \ell)$ iff $\hat{V}(r, \ell) = \hat{V}(r', \ell)$. We therefore have:

Theorem 3.17: $(r, \ell) \models C_N\varphi$ iff $(r', \ell) \models \varphi$ for all r' satisfying $\hat{V}(r, \ell) = \hat{V}(r', \ell)$.

Consequently, our local construction completely characterizes the connected components of the similarity graph, and hence when facts become common knowledge.

A Test for Common Knowledge

We now consider how this characterization gives rise to a test for common knowledge that processors can compute locally.

From Theorem 3.17, it follows that $\hat{V}(r, \ell)$ in a precise sense summarizes and uniquely determines the set of facts that are common knowledge at any given point (r, ℓ). The identity of $\hat{V}$ has two components: the failure pattern and input pattern during some prefix of the run. The fact that $\hat{V}$ becomes common knowledge implies that certain information about the failure pattern must become common knowledge. While it is the failure pattern alone that determines what views are contained in $\hat{V}$, it is difficult to characterize what properties of the failure pattern lead to these views being chosen by the construction, and hence what kinds of facts about the failure pattern become common knowledge. On the other hand, information about the input that follows from the views in $\hat{V}$ *does* characterize in a crisp way what facts about the input are common knowledge. Furthermore, it is easy to deduce from $\hat{V}$ whether the *existence* of a failure is common knowledge. As the following corollary will show, Theorem 3.17 implies that facts about the input and existence of failures that are common knowledge at the point (r, ℓ) must follow directly from the set $\hat{V}(r, \ell)$. We now make this statement precise. A run r, a set of processors G, and a time k determine a joint view $V = r_G(k)$. We denote by "V" the property of being a run in which the processors in G have the joint view V at time k (notice that G and k are uniquely determined by V). In other words, $(r, k) \models V$ iff $r_G(k) = V$. Thus, if $V \supset \varphi$ is valid in the system, then every run r' satisfying $r'_G(k) = V$ must also satisfy φ. We now have:

Corollary 3.18: Let φ be a fact about the input and the existence of failures, and let $V = \hat{V}(r,\ell)$. Then $(r,\ell) \models C_{\mathcal{N}}\varphi$ iff $V \supset \varphi$ is valid in the system.

Proof: Suppose $V \supset \varphi$ is valid in the system. By Lemma 3.14, we have $\hat{V}(r,\ell) = \hat{V}(r',\ell)$ for all runs r' such that $(r,\ell) \sim (r',\ell)$, and hence that $(r',\ell) \models V$ for all such r'. Given that $V \supset \varphi$ is valid in the system, we have $(r',\ell) \models \varphi$ for all such r'. It follows that $(r,\ell) \models C_{\mathcal{N}}\varphi$.

For the other direction, suppose that $V \supset \varphi$ is not valid in the system. Since $V \supset \varphi$ is not valid in the system, let u be a run such that $(u,\ell) \models V$ and yet $(u,\ell) \not\models \varphi$. We will construct a run s such that $(r,\ell) \sim (s,\ell)$, s and u have the same input, and s and u are the same with respect to the existence of failures (i.e., s will be failure-free iff u is). Since φ is a fact about the input and the existence of failures, $(u,\ell) \not\models \varphi$ will imply $(s,\ell) \not\models \varphi$. Since, in addition, $(r,\ell) \sim (s,\ell)$, we will have that $(r,\ell) \not\models C_{\mathcal{N}}\varphi$.

We construct s in two steps. We first construct a run v with the input of u satisfying $(r,\ell) \sim (v,\ell)$. Let v be the run with the failure pattern of r and the input of u. Given that r and v have the same failure pattern, and that $\hat{G}$ and $\hat{k}$ depend only on the failure pattern, we have that $\hat{G}(r,\ell) = \hat{G}(v,\ell)$ and $\hat{k}(r,\ell) = \hat{k}(v,\ell)$. Let us denote these values by $\hat{G}$ and $\hat{k}$. Since $(u,\ell) \models V$, we have $v(\hat{G},r,\hat{k}) = v(\hat{G},u,\hat{k})$, and hence $\mathcal{G}_{\hat{G}}(r,\hat{k}) = \mathcal{G}_{\hat{G}}(u,\hat{k})$. Since v and r have the same failure pattern, the *unlabeled* graphs underlying $\mathcal{G}_{\hat{G}}(v,\hat{k})$ and $\mathcal{G}_{\hat{G}}(r,\hat{k})$ (and hence also $\mathcal{G}_{\hat{G}}(u,\hat{k})$) are the same. Furthermore, since v and u have the same input, it follows that $\mathcal{G}_{\hat{G}}(v,\hat{k})$ and $\mathcal{G}_{\hat{G}}(u,\hat{k})$ (and hence also $\mathcal{G}_{\hat{G}}(r,\hat{k})$) are equal. Since $\mathcal{G}_{\hat{G}}(r,\hat{k}) = \mathcal{G}_{\hat{G}}(v,\hat{k})$ implies $\hat{V}(r,\ell) = \hat{V}(v,\ell)$, we have $(r,\ell) \sim (v,\ell)$ by Lemma 3.16.

We now consider the existence of failures, and construct the desired run s. If there is a failure in u, then let s be a run differing from v only in that a processor fails after time ℓ in s. Clearly $(v,\ell) \sim (s,\ell)$, and hence $(r,\ell) \sim (s,\ell)$. Conversely, if u is failure-free, then let $s = u$. Since u is failure-free, no processor in $\hat{G}$ knows of a failure at time $\hat{k}$ in u. Since processors in $\hat{G}$ have the same local state at time $\hat{k}$ in both u and r, the same is true of r. It follows that $\hat{B} = B(\hat{G},r,\hat{k})$ is empty, and since $\hat{G} = P - \hat{B}$, we have that $\hat{G} = P$. Notice that s differs from v only in that processors in $\hat{G} = P$ do not fail in s, and hence that $(v,\ell) \sim (s,\ell)$ by Lemma 3.15. Therefore, $(r,\ell) \sim (s,\ell)$. In either case, $(r,\ell) \sim (s,\ell)$, s and u have the same input,

and are the same with respect to the existence of failures. It follows by the above discussion that $(r, \ell) \not\models C_{\mathcal{N}}\varphi$. □

Corollary 3.18 summarizes the sense in which the construction allows us to test whether relevant facts are common knowledge at a given point. Let us consider the computational complexity of performing such tests. The first step in applying Corollary 3.18 to determine whether a fact is common knowledge at (r, ℓ) is to construct $\hat{V}(r, \ell)$. Recall that a group of processors implicitly knows that a processor is faulty iff it knows of a message the processor failed to send. This is an easy fact to check given the communication graph corresponding to the group's view. It follows that computing every iteration of the construction can easily be done in polynomial time. Furthermore, since the construction is guaranteed to converge within $t+1$ iterations, it follows that $\hat{G}$ and $\hat{k}$, and hence also $\hat{V}$ can be computed locally in polynomial time (as long as $\hat{V}$ is of polynomial size). Recall that if φ is a practical fact, then it is possible to determine in polynomial time whether or not $V \supset \varphi$ is valid in the system. Thus, given a practical simultaneous choice problem C, one polynomial-time implementation of a test for common knowledge of *enabled*(a_i) is to construct the set $V = \hat{V}$ and determine whether $V \supset$ *enabled*(a_i) is valid in the system. As a result, Theorem 3.4 implies the following:

Theorem 3.19: If C is an implementable, practical simultaneous choice, then there is a polynomial-time optimal protocol for C.

Discussion

We reiterate the fact that the resulting protocol for C is optimal *in all runs*: actions are performed in runs of $\mathcal{F}_C$ as soon as they could possibly be performed in runs of any other protocol, given the operating environment of the run. Thus, for example, simultaneous Byzantine agreement is performed in anywhere between 2 and $t+1$ rounds, depending on the pattern of failures (as is shown in [DM86] to be the case in the crash failure model). Similarly, the firing squad problem can be performed in anywhere between 1 and $t+1$ rounds after a "start" signal is received. Paradoxically, in all these cases, the simultaneous actions can be performed quickly only when many failures become known to the nonfaulty processors. In particular, if there are no

failures, no fact about the input is common knowledge less than $t+1$ rounds after it is first determined to hold.

Recall that every processor, faulty or nonfaulty, is able to compute the set $\hat{V}(r,\ell)$ locally. As a result, the following proposition shows that a fact is common knowledge to the nonfaulty processors iff it is common knowledge to all processors.

Proposition 3.20: Let φ be an arbitrary fact. In the omissions model, $C_{\mathcal{N}}\varphi \equiv C_{\mathcal{P}}\varphi$ is valid in all systems running a full-information protocol.

Proof: By Theorem 2.3, it is enough to show that $(r,\ell) \overset{\mathcal{P}}{\sim} (r',\ell)$ iff $(r,\ell) \overset{\mathcal{N}}{\sim} (r',\ell)$ for all runs r and r' and times ℓ. The 'if' direction is trivial, since $\mathcal{N} \subseteq \mathcal{P}$. The proof of the other direction is identical to the proof of Lemma 3.14, interpreting $\sim$ as $\overset{\mathcal{P}}{\sim}$. □

Proposition 3.20 implies that all processors (even the faulty processors) know exactly what actions are commonly known to be enabled in runs of $\mathcal{F}_c$. Thus, in this model the protocol $\mathcal{F}_c$ is guaranteed to satisfy a stronger version of simultaneous choice problems, in which condition (ii) is replaced by

(ii′) if a_i is performed by *any* processor (faulty or nonfaulty), then it is performed by *all* processors simultaneously.

Furthermore, since when an action is performed it is performed simultaneously by all processors, and since no other action is ever performed, there is no need for processors to continue sending messages after performing actions in runs of $\mathcal{F}_c$ in this model. We can therefore further reduce the communication of $\mathcal{F}_c$ by having processors halt after performing a simultaneous action. As a result, the following is an optimal protocol for any implementable simultaneous choice problem $\mathcal{C}$, an optimal protocol simpler than the protocol $\mathcal{F}_c$:

repeat *every round*
  send current local state to every processor
until $C_{\mathcal{N}}$*enabled*(a_i) *holds for some* a_i;
$j \leftarrow \min\{i : C_{\mathcal{N}}\textit{enabled}(a_i) \textit{ holds}\}$;
perform a_j;
halt.

The fact that in the omissions model the information in $\hat{V}(r,\ell)$ is essentially all that is common knowledge at a given point has interesting implications about the type of simultaneous actions that can be performed in this model. For example, recall that in the traditional simultaneous Byzantine agreement or consensus problems (see [PSL80, Fis83, DM86]), the processors are only required to decide, say, v in case they all start with an initial value of v. A stronger (and arguably more natural, or at least democratic) requirement, however, would require they decide v whenever the majority of initial values are v. This is clearly impossible, since some processors may be silent throughout the run. However, consider a protocol for simultaneous Byzantine agreement which is similar to $\mathcal{F}_c$, except that when some *enabled*(a_i) becomes common knowledge (which happens exactly when $\hat{V}$ becomes nonempty), the processors choose the value that appears in the majority of the initial values recorded in $\hat{V}(r,\ell)$ as their decision value. In this case, the processors actually approximate majority fairly well: If more than $(n+t)/2$ of the initial values are v, then v will be chosen. In fact, we can show that the approximation is bad only in runs in which agreement is obtained early. In particular, if agreement cannot be obtained before time $t+1$ (this would happen in runs r for which $\hat{V}(r,\ell)$ contains only empty local states for every $\ell \leq t$), then the value agreed upon would be the majority value in case more than $(n+1)/2$ of the processors have the same initial value. Furthermore, a protocol for *weak* (exact) majority *does* exist: A protocol that either decides that there was a failure or decides on the true majority value.

Since messages from faulty processors can convey new information about the failure pattern, such messages *do* affect the construction. Therefore, the behavior of faulty processors, even after they have been discovered to be faulty, plays an important role in determining what facts become common knowledge and when. In the crash failure model, however, a failed processor does not communicate with other processors after its failing round and has little impact on what facts become common knowledge. This is an essential property of the omissions model distinguishing it from the crash failure model.

We note, however, that all of the analysis in this subsection applies to the crash failure model, with all of the proofs applying verbatim when restricted to the crash failure model. We thus have:

Proposition 3.21: In the crash failure model, $(r,\ell) \models C_{\mathcal{N}}\varphi$ iff it is the case

that $(r', \ell) \models \varphi$ for all r' satisfying $\hat{V}(r, \ell) = \hat{V}(r', \ell)$.

Thus, the set $\hat{V}(r, \ell)$ completely characterizes what facts are common knowledge at the point (r, ℓ) in the crash failure model as well. Since the same proofs show that the construction characterizes the connected components of the similarity graph in both the omissions and the crash failure model, the similarity graph in the omissions model is simply an extension of the similarity graph in the crash failure model obtained by adding nodes and edges to the similarity graph in the crash failure model, not breaking up the connected components appearing in the crash failure model. This implies that in a run of the omission model having a failure pattern consistent with the crash failure model, exactly the same facts about the input and the existence of failures are common knowledge at any given time in both the crash failure and the omissions model. (However, as a result of the difference in the types of failures possible in the two failure models, different facts about the failure pattern are common knowledge at the corresponding points.) Ruben Michel has independently characterized the similarity graph in variants of the crash failure model (see [Mic88]). For the crash failure model itself, he has an alternative construction that also characterizes the connected components of the similarity graph.

As in the omissions model, it follows from Proposition 3.21 that our construction can be used to derive efficient optimal protocols for simultaneous choice problems in the crash failure model, thus showing that results similar to those proven in [DM86] in the crash model can be obtained in the omissions model, although our techniques are quite different. We therefore have the following:

Corollary 3.22: Let $\mathcal{C}$ be an implementable, practical simultaneous choice. In the crash failure model, there is a polynomial-time optimal protocol for $\mathcal{C}$.

As a final remark, let k_i and G_i be the intermediate results of beginning the construction at the point (r, ℓ), and denote $v(G_i, r, k_i)$ by V_i. Consider the operator $\mathcal{E}$ defined by $\mathcal{E}(V_i) = V_{i+1}$ for all i. We find it interesting that $\hat{V}$, which is a fixed point of the operator $\mathcal{E}$, characterizes the facts φ for which $C_{\mathcal{N}}\varphi$ holds, where we know from [HM84] that $C_{\mathcal{N}}\varphi$ is a fixed point of $E_{\mathcal{N}}$ (see Proposition 2.5). While researchers are used to thinking semantically of common knowledge as a fixed point, this construction shows how we can think combinatorially of common knowledge as a fixed point, as well.

3.5.2 Receiving Omissions

In the omissions model, faulty processors fail only to *send* messages. In this subsection, we consider the symmetric *receiving omissions* model, in which faulty processors fail only to *receive* messages. While at first glance these models seem quite similar, they are actually extremely different. In particular, we will see that testing for common knowledge in this model becomes trivial. As a result, efficient optimal protocols for practical simultaneous choice problems become completely trivial in this model.

One intriguing difference between the omissions model and the receiving omissions model is the following. We have seen in the omissions model that in some cases a fact (for example, the arrival of a "start" signal) does not become common knowledge until as many as $t+1$ rounds after it is first determined to hold. Intuitively, the attainment of common knowledge is delayed by the possibility that a processor might fail to send a message determining that the fact holds. However, in the receiving omission model even *faulty* processors send all messages required by the protocol. Since nonfaulty processors receive all messages sent to them, in runs of a full-information protocol all nonfaulty processors have a complete view of the first k rounds at time $k+1$. We can thus show the following:

Theorem 3.23: Let φ be a fact about the first k rounds. In the receiving omissions model, $(r,k) \models \varphi$ iff $(r,k+1) \models C_N\varphi$.

The proof of this result depends on the notion of a fact being valid at time k: A fact φ is said to be *valid* (in the system) *at time* k if for all runs r we have $(r,k) \models \varphi$. We remark that the following variant of the induction rule holds:

If	$\varphi \supset E_s\varphi$	is valid at time k,
then	$\varphi \supset C_s\varphi$	is valid at time k.

Proof: Since φ is a fact about the first k rounds, $(r,k) \models \varphi$ iff $(r,k+1) \models \varphi$. Thus, it is enough to show that $(r,k+1) \models \varphi$ iff $(r,k+1) \models C_N\varphi$. Notice that $(r,k+1) \models C_N\varphi$ implies $(r,k+1) \models \varphi$. Conversely, suppose $(r,k+1) \models \varphi$. During round $k+1$ in r every processor sends its entire local state to all processors, so at time $k+1$ all nonfaulty processors have a complete view of the first k rounds of r. Since φ is a fact about the first k rounds, $(r,k+1) \models E_N\varphi$. We have just shown that $\varphi \supset E_N\varphi$ is valid at

time $k+1$, so $\varphi \supset C_{\mathcal{N}}\varphi$ is valid at time $k+1$ as well. Thus, $(r, k+1) \models \varphi$ implies $(r, k+1) \models C_{\mathcal{N}}\varphi$. □

As a consequence of Theorem 3.23, polynomial-time optimal protocols for practical simultaneous choice problems are very simple in this model. Again, by polynomial-time here we will mean polynomial in n, t, and the round number ℓ.

Corollary 3.24: Let $\mathcal{C}$ be an implementable, practical simultaneous choice. In the receiving omissions model, there is a polynomial-time optimal protocol for $\mathcal{C}$.

Proof: Since $\mathcal{C}$ is implementable, Theorem 3.4 implies that $\mathcal{F}_{\mathcal{C}}$ is an optimal protocol for $\mathcal{C}$. It remains to show that $\mathcal{F}_{\mathcal{C}}$ can be implemented in polynomial time. Since the messages sent by $\mathcal{F}_{\mathcal{C}}$ can clearly be computed in polynomial time, we need only show how to implement the tests for common knowledge of the conditions $enabled(a_i)$ in polynomial time. We claim that $(r, \ell) \models C_{\mathcal{N}} enabled(a_i)$ iff $\mathcal{G}(r, \ell - 1) \supset enabled(a_i)$ is valid in the system. Since $\mathcal{C}$ is a practical simultaneous choice problem, determining whether $\mathcal{G}(r, \ell - 1) \supset enabled(a_i)$ is valid in the system can be done in polynomial time. As $\mathcal{G}(r, \ell - 1)$ can be determined by all nonfaulty processors at (r, ℓ) in polynomial-time, this will yield a polynomial-time implementation of a test for common knowledge of $enabled(a_i)$, and we will be done. Suppose $\mathcal{G}(r, \ell - 1) \supset enabled(a_i)$ is valid in the system. Theorem 3.23 implies that $\mathcal{G}(r, \ell - 1)$ is common knowledge at (r, ℓ), and it follows that $(r, \ell) \models C_{\mathcal{N}} enabled(a_i)$. Conversely, suppose $(r, \ell) \models C_{\mathcal{N}} enabled(a_i)$. Let s be a run satisfying $\mathcal{G}(r, \ell - 1)$. A proof similar to the base case of Lemma 3.8 shows that $(r, \ell) \sim (s, \ell)$. Since $(r, \ell) \models C_{\mathcal{N}} enabled(a_i)$, it follows that $(s, \ell) \models enabled(a_i)$. Thus, $\mathcal{G}(r, \ell - 1) \supset enabled(a_i)$ is valid in the system, as desired. □

The results of this section point out a number of interesting differences between the omissions model and the receiving omissions model. For example, consider the distributed firing squad problem. First, Theorem 3.23 implies that all nonfaulty processors are able to fire in the receiving omission model exactly one round after the first "start" signal is received. Recall that in the omissions model, firing may delayed as many as $t+1$ rounds. Second, since a faulty processor p might fail to receive all messages, it is not possible to guarantee that p will ever fire following the receipt of a "start" signal by a nonfaulty processor. In the omissions model we have shown that it is

possible to guarantee that *all* processors perform any action (e.g., "firing") performed by the nonfaulty processors. Finally, notice that faulty processors may sometimes be unable to halt, or terminate their participation in a distributed firing squad protocol, even long after the nonfaulty processors have fired: A processor p receiving no messages or "start" signals at all can never halt since at any point it is possible (according to p's local state) that it will be the only processor in the system to receive a "start" signal. In this case, *optimal* protocols must require the nonfaulty processors to fire one round later, and hence p must be able to send this information to the nonfaulty processors. In contrast, in the omissions model it is possible to guarantee that all processors halt as soon as an action is performed in the system. These remarks show that while at first glance the assignment of responsibility for undelivered messages to sending or to receiving processors may seem arbitrary, the assignment has a dramatic effect on when facts become common knowledge, and hence on the behavior of optimal protocols. Since such a simple modification of the omissions model results in the collapse of the combinatorial structure underlying the model (witness Theorem 3.23), we consider this to be an indication that the omissions model is not a robust model of failure.

3.5.3 Generalized Omissions

We have just seen that the choice of whether sending or receiving processors are responsible for undelivered messages has a dramatic effect on the structure of the omissions model. Consider, however, the *generalized omissions* model, in which a faulty processor may fail both to send *and* to receive messages. This section is concerned with the design of optimal protocols for simultaneous choice problems in this model. We have seen that Theorem 3.4 implies the protocol $\mathcal{F}_c$ is an optimal protocol in this model, and that Theorem 3.7 implies this protocol can be implemented in polynomial-space. As in previous sections, the remaining question is whether there are efficient optimal protocols in this model. The principal result of this section is that testing for common knowledge in the generalized omissions model is NP-hard. Using the close relationship between common knowledge and simultaneous actions, we obtain as a corollary that optimal protocols for most any simultaneous choice problem in this model require processors to perform NP-hard computations. Consequently, for example, in this model there can

be no efficient optimal protocol for simultaneous Byzantine agreement or the distributed firing squad problem. This is a dramatic difference between the generalized omissions model and the more benign failure models, where, as we have seen, efficient optimal protocols *do* exist.

One important difference between the generalized omissions model and simpler variants of the omissions model is that in the generalized omissions model undelivered messages do not necessarily identify the set of faulty processors, but merely place constraints on their possible identities: Either the sender or the intended receiver of every undelivered message must be faulty. The faulty processors must therefore induce a "vertex cover" of the undelivered messages. Recall that in our analysis of the omissions failure model, determining the number and the identity of the faulty processors given the labeled communication graph of a point played a crucial role in characterizing the facts that are common knowledge at a point. In that model, a processor is known to be faulty iff it is known that a message it was supposed to send was not delivered, a fact easily determined from the labeled communication graph. In the generalized omissions model, however, even determining the number (and not necessarily the identities) of processors implicitly known to be faulty essentially involves computing the size of the minimal vertex cover of a graph, a problem known to be NP-complete (see [GJ79]). It is with this intuition that we now proceed to show that determining whether certain facts are common knowledge is computationally prohibitive in the generalized omissions model, assuming P$\neq$NP.

However, in order to study the complexity of testing for common knowledge in the generalized omissions model in a meaningful way, we are once again faced with the need to restrict our attention to a class of facts that includes all of the facts that may arise in natural simultaneous choice problems, and excludes anomalous cases. For example, if φ is valid in the system, then so is $C_N\varphi$, and testing whether φ is common knowledge is a trivial task. On the other hand, one can imagine facts involving excessive computational complexity of a type irrelevant to simultaneous choice problems. Consider, for instance, a fact φ with the property that the communication graph of any point satisfying φ encodes information allowing the solution of all problems in NP of size smaller than the number of processors in the system. Whereas it seems unlikely that such a fact exists, such a statement is probably very hard to prove, and it is definitely not the business of this chapter to do so.

We are therefore led to make the following restriction. A fact φ is said to

be *admissible* within a class of systems running a full-information protocol if (i) for all systems within this class neither φ nor $\neg\varphi$ is valid in the system, and (ii) there is a polynomial-time algorithm explicitly constructing for each system a labeled communication graph $\mathcal{G}(r,\ell)$ of minimal length having the property that $\mathcal{G}(r,\ell) \supset \varphi$ is valid in the system. Condition (i) simply says that in none of these systems is testing for φ completely trivial. Condition (ii) says that in each of these systems it has to be easy to generate enough of a communication graph to guarantee that φ is true at any point of any run with this communication graph. The ability to generate such a graph will be used to generate the graph we submit to a given test for common knowledge of φ. We say that a simultaneous choice problem C is admissible if each condition *enabled*(a_i) is admissible within the class of systems determined by a full-information protocol and C. We claim that any natural simultaneous choice is admissible. We can now state the fundamental result of this section which says, loosely speaking, that testing for common knowledge of admissible facts $\varphi_1, \ldots, \varphi_b$ is NP-hard.

For given facts $\varphi_1, \ldots, \varphi_b$ $(b \geq 1)$ and a class $\Sigma = \{\Sigma(n,t) : n \geq t+2\}$ of systems, define the decision problem of *testing for common knowledge of* $\varphi_1, \ldots, \varphi_b$ *in* Σ as follows: Given as input a graph $\mathcal{G}_i(r,\ell)$ corresponding to p_i's local state at a point (r,ℓ) of a system in Σ with $n > 2t$,[6] does $(r,\ell) \models \bigvee_i C_N \varphi_i$?

Lemma 3.25: Let $\varphi_1, \ldots, \varphi_b$ be admissible, practical facts within a class Σ of systems running a full-information protocol in the generalized omissions model. The problem of testing for common knowledge of $\varphi_1, \ldots, \varphi_b$ in Σ is NP-hard (in n).

The proof of Lemma 3.25 will follow shortly. Notice, however, that t is variable in the statement of this lemma, and in general may be $O(n)$. The proof of this result will not apply for a fixed t, nor to cases in which t is restricted, say, to be $O(\log n)$. In any case, it will follow that any standard implementation of our optimal knowledge-based protocols must be computationally intractable, unless P=NP. It is natural to ask whether this inefficiency is merely the result of having programmed our protocols using tests

[6] We note that the condition $n > 2t$ seems odd, but this slightly stronger formulation of testing for common knowledge is needed later when proving the intractability of optimal protocols for simultaneous choice problems in this model.

for common knowledge. It is conceivable, for instance, that there are optimal protocols for admissible simultaneous choice problems in the generalized omissions model that *are* computationally efficient. Intuitively, however, in order to perform a simultaneous action, an optimal protocol $\mathcal{P}$ must essentially determine whether any of the conditions *enabled*(a_i) is common knowledge. Corollary 3.3 implies that such a condition becomes common knowledge during the corresponding run of a full-information protocol as soon as it does during a run of $\mathcal{P}$. Thus, an optimal protocol $\mathcal{P}$ must essentially determine whether such a fact is common knowledge during the corresponding run of a full-information protocol $\mathcal{F}$. Since Lemma 3.25 implies that this problem is NP-hard, computing the function $\mathcal{P}$ must be NP-hard as well. We now make this argument precise.

Recall that a protocol is formally a function mapping n, t, and a processor's state to a list of the actions the processor should perform, followed by a list of the messages it is required to send in the following round. We say that a protocol is *communication-efficient* if in a system of n processors the size of the messages each processor is required to send during round ℓ is polynomial in n and ℓ. In the following result we show that the problem of computing the function corresponding to a communication-efficient optimal protocol for a simultaneous choice problem is NP-hard. Hence, no such protocol can be computationally efficient, unless P=NP.

For a given protocol $\mathcal{P}$ and class $\Sigma = \{\Sigma(n,t) : n \geq t+2\}$ of systems, define the problem of *computing* $\mathcal{P}$ *in* Σ as follows: Given as input a graph $\mathcal{G}_i(r,\ell)$ corresponding to p_i's local state at a point (r,ℓ) of a system in Σ, output the list of messages p_i is required by $\mathcal{P}$ to send at (r,ℓ), and output the list of actions p_i is required by $\mathcal{P}$ to perform at (r,ℓ).

Theorem 3.26: Let $\mathcal{C}$ be an admissible, practical simultaneous choice with actions $a_1,\ldots,a_b$, and let $\mathcal{P}$ be a communication-efficient, optimal protocol for $\mathcal{C}$. Let Σ be the class of systems determined by $\mathcal{P}$ and $\mathcal{C}$. There is a Turing reduction from the problem of testing for common knowledge of *enabled*$(a_1),\ldots,$ *enabled*(a_b) in Σ to the problem of computing $\mathcal{P}$ in Σ. In this sense, the problem of computing $\mathcal{P}$ in Σ is NP-hard (in n).

Proof: Notice that since $\mathcal{P}$ is a protocol for $\mathcal{C}$, the problem $\mathcal{C}$ must be implementable, and Theorem 3.4 implies that the full-information protocol $\mathcal{F}_c$ must be an optimal protocol for $\mathcal{C}$. Let $\Sigma = \{\Sigma(n,t) : n \geq t+2\}$ be the

class of systems determined by $\mathcal{C}$ and $\mathcal{F}_c$. Since $\mathcal{C}$ is an admissible, practical simultaneous choice, each condition *enabled*(a_i) must be an admissible, practical fact within Σ. By Lemma 3.25, given the graph $\mathcal{G}(r,\ell)$ of a point (r,ℓ) in a system $\Sigma(n,t)$ with $n > 2t$, the problem of determining whether $(r,\ell) \models \bigvee_i C_{\mathcal{N}}\mathit{enabled}(a_i)$ is NP-hard. We will exhibit a Turing reduction from this problem to the problem of computing $\mathcal{P}$; that is, given the graph $\mathcal{G}(r,\ell)$ of a point (r,ℓ) in a system $\Sigma(n,t)$ with $n > 2t$, we will show how to use $\mathcal{P}$ to determine in polynomial time whether $(r,\ell) \models \bigvee_i C_{\mathcal{N}}\mathit{enabled}(a_i)$. Having exhibited such a reduction, we will have shown that the problem of computing $\mathcal{P}$ is NP-hard.

Let r be a run of $\mathcal{F}_c$ in a system $\Sigma(n,t)$ with $n > 2t$, and let s be the corresponding run of $\mathcal{P}$. It follows from the definition of $\mathcal{F}_c$ that $(r,\ell) \models \bigvee_i C_{\mathcal{N}}\mathit{enabled}(a_i)$ iff the nonfaulty processors perform a simultaneous action no later than time ℓ in r. Since $\mathcal{F}_c$ and $\mathcal{P}$ are both optimal protocols for $\mathcal{C}$, the nonfaulty processors perform simultaneous actions at the same times during r and s. Since $n > 2t$, there must be at least $t+1$ nonfaulty processors in both runs, so the nonfaulty processors simultaneously perform an action no later than time ℓ in either run iff $t+1$ processors do so. Therefore, $(r,\ell) \models \bigvee_i C_{\mathcal{N}}\mathit{enabled}(a_i)$ iff $t+1$ processors perform a simultaneous action no later than time ℓ in s.

One algorithm for determining whether $t+1$ processors *do* perform a simultaneous action no later than time ℓ in s is to construct the local state of each processor in s at each time k before time ℓ, and use $\mathcal{P}$ to determine when processors are required to perform actions. Suppose we have constructed the state of each processor at time $k-1$ in s; let us consider the problem of constructing the state of a processor p at time k. Processor p's state at (s,k) consists of p's name, the time k, a list of the messages received by p during the first k rounds of s, and a list of the input received by p during the first k rounds of s. Recall that since r is a run of a full-information protocol, the graph $\mathcal{G}(r,\ell)$ is actually an encoding of the operating environment during the first ℓ rounds of r, and hence also of s. Given the states of all processors at time $k-1$, the protocol $\mathcal{P}$ determines what message each processor is required to send to p, and $\mathcal{G}(r,\ell)$ determines which of these messages are actually delivered to p in s. Since $\mathcal{P}$ is communication-efficient, each of these messages is of size polynomial in n and k. Furthermore, the input received by p during round k labels the node $\langle p,k\rangle$ of $\mathcal{G}(r,\ell)$. Since $\mathcal{C}$ is practical, this input is of constant size. Thus, given each processor's state at time $k-1$,

we can use the graph $\mathcal{G}(r,\ell)$ and an oracle for $\mathcal{P}$ to construct the state of each processor at time k of s in polynomial time. (An oracle for $\mathcal{P}$ is an oracle that, given the state of a processor p at a point (r,ℓ), in one step determines what actions $\mathcal{P}$ requires p to perform at time ℓ, and constructs the messages $\mathcal{P}$ requires p to send during round $\ell+1$.)

Consider the following algorithm:

```
action_performed ← false;
k ← 0;
repeat
   for all processors p do
      determine whether P requires p to perform any action at time k, and
      construct the messages P requires p to send during round k + 1;
   endfor
   if t + 1 processors perform actions at time k
      then action_performed ← true;
   k ← k + 1;
until k > ℓ or action_performed;
if action_performed
   then halt with "yes"
   else halt with "no".
```

From the previous discussion it is clear that given any oracle for $\mathcal{P}$, this algorithm determines in polynomial time whether $t+1$ processors perform actions simultaneously no later than time ℓ in s, and hence whether $(r,\ell) \models \bigvee_i C_N enabled(a_i)$. □

As an immediate corollary of Theorem 3.26, we have the following:

Corollary 3.27: Let $\mathcal{C}$ be an admissible practical simultaneous choice problem. If there is a polynomial-time optimal protocol for $\mathcal{C}$, then P=NP.

Corollary 3.27 implies that optimal protocols for simultaneous choice problems as simple as the distributed firing squad problem or simultaneous Byzantine agreement are computationally infeasible in the generalized omissions model, assuming P $\neq$ NP. In fact, we do not know whether these problems can be implemented in polynomial time even using an NP oracle. The best we can do in the generalized omissions model is implement them using polynomial-space computations, as in the proof of Theorem 3.7. We

consider the question of determining the exact complexity of implementing admissible practical simultaneous choice problems in this model an interesting open problem.

We now proceed to prove Lemma 3.25. First, however, we state a result that will be very useful in the proof of Lemma 3.25. Roughly speaking, it says that if a group of processors can (jointly) prove that they are nonfaulty, then their states become common knowledge at the end of the following round.

Lemma 3.28: Let S be a set of processors and let $\overline{S} = P - S$. Let r be a run of a full-information protocol. If the processors in S implicitly know at $(r, \ell - 1)$ that $\overline{S}$ contains t faulty processors, then the joint view of S at $(r, \ell - 1)$ is common knowledge at (r, ℓ).

Proof: Let φ = "V is the joint view of S at time $\ell - 1$", where $V = v(S, r, \ell - 1)$. Suppose $(r', \ell) \models \varphi$. Given that S has the same joint view at $(r, \ell - 1)$ and at $(r', \ell - 1)$, and since S implicitly knows at $(r, \ell - 1)$ that $\overline{S}$ contains t faulty processors, S implicitly knows the same at $(r', \ell - 1)$. In particular, the processors in S must be nonfaulty in r', and each must successfully send its state to all processors during round ℓ of r'. Since all nonfaulty processors will receive these messages, we have $(r', \ell) \models E_N\varphi$. It follows that $\varphi \supset E_N\varphi$ is valid at time ℓ, and the induction rule implies $\varphi \supset C_N\varphi$ is valid at time ℓ as well. Thus, $(r, \ell) \models \varphi$ implies $(r, \ell) \models C_N\varphi$. □

(We note in passing that a converse to Lemma 3.28 is also true: If the joint view at time $\ell - 1$ of a set S of processors is common knowledge at time ℓ, then the processors in some set $S' \supseteq S$ must implicitly know at time $\ell - 1$ that there are t faulty processors among the members of $\overline{S}'$.)

In addition to Lemma 3.28, the following result, analogous to Lemma 3.8 in the omissions model, will be of use in the proof of Lemma 3.25.

Lemma 3.29: Let r and r' be runs differing only in the (faulty) behavior displayed by processor p after time k, and suppose no more that f processors fail in either r or r'. If $\ell - k \leq t + 1 - f$, then $(r, \ell) \sim (r', \ell)$.

Proof: The proof is analogous to the proof of Lemma 3.8, with the added observation that if p sends no messages after (an arbitrary) time k' in s, then $(s, \ell) \sim (s', \ell)$ where s' differs from s in that p receives messages from an arbitrary set of processors during round k'. □

Finally, as previously mentioned, the proof of Lemma 3.25 involves a reduction from the *Vertex Cover* problem. This is the problem (see [GJ79]) of determining, given a graph $G = (V, E)$ and a positive integer M, whether G has a *vertex cover* of size M or less; that is, a subset $\mathcal{V} \subseteq V$ such that $|\mathcal{V}| \leq M$ and, for each edge $\{v, w\} \in E$, at least one of v or w belongs to $\mathcal{V}$.

Theorem [Karp]: *Vertex Cover* is NP-complete.

We now prove Lemma 3.25.

Proof of Lemma 3.25: We will exhibit a Turing reduction from Vertex Cover to the problem of testing for common knowledge of $\varphi_1, \ldots, \varphi_b$, and it will follow that this problem is NP-hard.

Since every graph $G = (V, E)$ is $|V|$-coverable, the following is an algorithm for *Vertex Cover*:

$m \leftarrow |V|$;
while *G has a vertex cover of size $m - 1$* **do**
  $m \leftarrow m - 1$;
if $m \leq M$
  then return *"G has a vertex cover of size M"*
  else return *"G has no vertex cover of size M"*.

To implement this test, it is enough to implement a test that, given an m-coverable graph G, determines whether G is $(m-1)$-coverable. Every graph $G = (V, E)$ clearly has a vertex cover of size $|V| - 1$. In addition, it is possible to determine whether G has a vertex cover of size $|V| - 2$ in polynomial time. Similarly, it is easy to determine whether G has a vertex cover of size 0 in polynomial time. We show that if $1 \leq m \leq |V| - 2$ and G is m-coverable, then it is possible to construct in polynomial time a graph $\mathcal{G}(r, \ell)$ with the property that $(r, \ell) \models \bigvee_i C_N \varphi_i$ iff G is *not* $(m-1)$-coverable. The point (r, ℓ) will be a point of a system $\Sigma(n, t)$ with $n > 2t$ from the class under consideration (i.e., the class of systems running a full-information protocol in the generalized omissions model). Thus, given an oracle for testing for common knowledge of $\varphi_1, \ldots, \varphi_b$, we will have a polynomial-time test for the $(m-1)$-coverability of G. It will follow that testing for common knowledge of $\varphi_1, \ldots, \varphi_b$ is NP-hard.

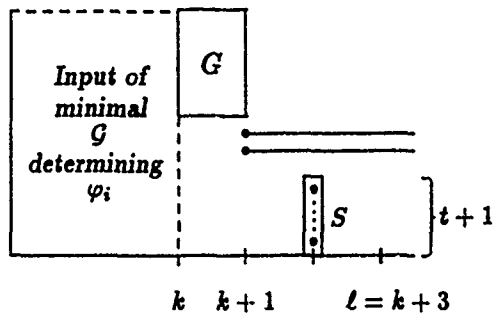


Figure 3.5: Embedding a graph G in a run r.

Fix a graph $G = (V, E)$ and an integer m satisfying $1 \leq m \leq |V| - 2$. Let $n = |V| + m + 3$ and $t = m + 2$, and let $\Sigma(n, t)$ be a system from the class under consideration. Notice that since $|V| \geq m + 2$, we have $n > 2t$. Since each fact φ_i is admissible, for each φ_i we can explicitly construct in polynomial time a labeled communication graph (of a point in $\Sigma(n, t)$) of minimal length determining φ_i. Of these graphs, let $\mathcal{G}$ be one of minimal length, say of length k. Let r be a run of $\Sigma(n, t)$, illustrated in Figure 3.5, satisfying the following conditions: (i) the input received in the first k rounds of r is the same as in $\mathcal{G}$, and no input is received after time k; (ii) all messages in the first k rounds are delivered; (iii) in round $k + 1$, the only undelivered messages are as follows: no message is delivered from processor p_v to p_w in round $k + 1$ of r iff there is an edge from v to w in G (that is, the graph G is represented by the undelivered messages during round $k + 1$); (iv) two additional processors f_1 and f_2 are silent from time $k + 1$ in r, and all other messages after time $k + 1$ are delivered; and (v) a set S of $t + 1$ additional processors do not fail in r. Since G has a vertex cover $\mathcal{V}$ of size m, one failure pattern consistent with the undelivered messages in r is that p_v is faulty for every $v \in \mathcal{V}$ (accounting for the undelivered messages during round $k+1$ of r) and that both f_1 and f_2 are faulty. Given that $t = m+2$ processors fail in this failure pattern, there is a run r of $\Sigma(n, t)$ satisfying the required conditions. Since the graph $\mathcal{G}$ determining the input of $\mathcal{G}(r, k)$ can be constructed in polynomial time, setting $\ell = k + 3$, the graph $\mathcal{G}(r, \ell)$ can be constructed in polynomial time as well. It remains to show that $(r, \ell) \models \bigvee_i C_N \varphi_i$ iff G is not $(m - 1)$-coverable.

Suppose G has no vertex cover of size $m-1$, and let F be the set of processors failing in r. Since f_1 and f_2 must be faulty (each fails to the $t+1$ processors in S), $F' \stackrel{\text{def}}{=} F - \{f_1, f_2\}$ must account for every undelivered message during round $k+1$. If there is an edge from v to w in G, then no message from p_v to p_w is delivered in round $k+1$, and one of p_v or p_w must be in F'. It follows that F' must induce a vertex cover of G. Since G has no vertex cover of size $m-1$, F' must contain at least m processors, and F at least $t = m+2$. Thus, the processors in S implicitly know at time $k+2$ that their complement $\overline{S} = P - S$ contains t faulty processors. By Lemma 3.28, their states at time $k+2$ must be common knowledge at time $k+3$. These states contain a complete description of $\mathcal{G}(r,k)$, and hence the identity of $\mathcal{G}(r,k)$ is common knowledge at (r,ℓ). Recall that $\mathcal{G}$ was chosen to be a graph determining φ_i for some i. If $\mathcal{G}$ does not specify a failure, then $\mathcal{G}(r,k) = \mathcal{G}$, and it follows that $(r,\ell) \models C_N\varphi_i$. On the other hand, if $\mathcal{G}$ *does* specify a failure, then φ_i is determined by the input to the first k rounds of $\mathcal{G}$ and the existence of a failure. Notice that the failure of f_1 and f_2 is also recorded in the view of S at time $k+2$, and hence is also common knowledge at (r,ℓ). Thus, both the input to the first k rounds of $\mathcal{G}$ and the existence of a failure are common knowledge at time ℓ, and it follows that $(r,\ell) \models C_N\varphi_i$. In either case, we have $(r,\ell) \models \bigvee_i C_N\varphi_i$.

Conversely, suppose G *does* have a vertex cover of size $m-1$. Without loss of generality, at most $t-1$ processors fail in r. First, we claim that $(r,\ell) \sim (s,\ell)$ where s is a failure-free run with the input of r. Since f_1 and f_2 fail only after time $k+1 = \ell-2$, two applications of Lemma 3.29 imply that $(r,\ell) \sim (r',\ell)$ where r' differs from r in that f_1 and f_2 do not fail in r'. Since at most $t-3$ processors fail in r' and $k = \ell-3$, by Lemma 3.29 we have $(r',\ell) \sim (s,\ell)$. Second, we claim that for each φ_i there is a run u_i not satisfying φ_i that differs from $\mathcal{G}$ only after time $k-1$. If $k=0$, then since φ_i is admissible and hence not valid in the system, such a run must certainly exist. On the other hand, if $k>0$, then since $\mathcal{G}$ was chosen to be a labeled communication graph of minimal length determining φ_j for some φ_j, such a run must exist in this case as well. Now, let u_i' be a run having the input of u_i, in which no processor fails before time ℓ, and in which processors become silent after time ℓ iff there is a failure in u_i. Since φ_i is a fact about the input and existence of failures, and since u_i does not satisfy φ_i, neither does u_i'. Let $\hat{s}$ and $\hat{u}_i'$ be runs of $\mathcal{F}$ in the omissions model having the

operating environments of s and u'_i, respectively. (Notice that these operating environments actually are operating environments of the omissions model.) Notice that no processor fails before time ℓ in either $\hat{s}$ or $\hat{u}'_i$. It follows that $\hat{G}(\hat{s},\ell) = \hat{G}(\hat{u}'_i,\ell)$, and that $\hat{k}(\hat{s},\ell) = \hat{k}(\hat{u}'_i,\ell)$. We denote these values by $\hat{G}$ and $\hat{k}$, respectively. Since $t = m+2$ and $m \geq 1$, we have that $t \geq 3$. Thus, $\hat{k} = \ell-(t+1) \leq \ell-4 = k-1$. Recall that $\hat{s}$ and $\hat{u}'_i$ have the same input (and no failures) through time $k-1$. It follows that $\hat{V}(\hat{s},\ell) = \hat{V}(\hat{u}'_i,\ell)$. It follows by Lemma 3.16 that $(\hat{s},\ell) \sim (\hat{u}'_i,\ell)$ in the omissions model, and hence that $(s,\ell) \sim (u'_i,\ell)$ in the generalized omissions model as well. Since $(r,\ell) \sim (s,\ell)$, it follows that for each φ_i we have $(r,\ell) \sim (u'_i,\ell)$ and $(u'_i,\ell) \not\models \varphi_i$. Therefore, for each φ_i we have $(r,\ell) \not\models C_{\mathcal{N}}\varphi_i$, and hence $(r,\ell) \not\models \bigvee_i C_{\mathcal{N}}\varphi_i$. □

We have seen that, as a result of the uncertainty about the failure pattern, the complexity of determining whether admissible facts are common knowledge is dramatically greater in this model than in more benign models. It is conceivable, however, that this gap in complexity is due to the fact that faulty processors may fail both to send and to receive messages, and not merely due to the uncertainty about the failure pattern. We can show, however, that it is *precisely* due to this uncertainty that we observe this complexity gap. Consider the closely related failure model we have termed ***generalized omissions with information***, a model differing from the generalized omissions model in that a processor not receiving a message can determine whether it or the sender is at fault. We can show that the construction used in the omissions model can also be used in this model to yield a set of states $\hat{V}(r,\ell)$ completely characterizing what facts are common knowledge at the point (r,ℓ).

Proposition 3.30: In generalized omissions with information, we have $(r,\ell) \models C_{\mathcal{N}}\varphi$ iff $(r',\ell) \models \varphi$ for all r' satisfying $\hat{V}(r',\ell) = \hat{V}(r,\ell)$.

All of the proofs in the omissions model hold when generalized to this model, with the exception that the construction must be started with a nonfaulty processor. (In particular, Lemma 3.13 holds only when the processors p and q are processors that do not fail to receive messages.) This exception says that faulty processors may not be able to perform all actions performed by the nonfaulty processors, but this is no surprise since the same is true in the receiving omissions model. Furthermore, the computation of the sets B_i in the construction now depends not only on the undelivered messages, but also on the additional information that receiving processors obtain regarding blame

for the undelivered messages. As in the omissions model, this construction yields a method of deriving efficient tests for common knowledge of certain facts. Thus, it is again possible to design efficient optimal protocols:

Theorem 3.31: Let C be an implementable practical simultaneous choice. In generalized omissions with information, there is a polynomial-time optimal protocol for C.

This shows that it is precisely the uncertainty about the failure pattern that is responsible for the observed gap in complexity, and not merely the fact that faulty processors may fail both by failing to send *and* to receive messages.

The uncertainty about the failure pattern in the generalized omissions model adds a new combinatorial structure to the similarity graph in this model that does not exist in other variants of the omissions model. Since it is possible to assign failure to processors in a number of different ways consistent with a pattern of undelivered messages, it is possible to play a solitaire version of a "pebbling game" with the failure pattern when constructing paths in the similarity graph, showing that one point is similar to another point by alternatively assigning responsibility for undelivered messages to the sender and to the receiver. In fact, in addition to increasing the difficulty of determining *whether* a fact is common knowledge at a point, this new combinatorial structure has interesting effects on *when* facts become common knowledge. Recall from the discussion at the end of Section 6.1 that the similarity graph in the omissions model is simply an extension of the similarity graph in the crash failure model, two points with crash failure patterns being similar in the crash failure model iff they are in the omissions model. As a result, our optimal protocol $\mathcal{F}_c$ in the omissions model is also an optimal protocol when restricted to runs of the crash failure model. In the generalized omissions model, however, the similarity graph is *not* merely an elaboration of the similarity graph in the omissions model: A connected component in the similarity graph of the generalized omissions model may contain several distinct connected components from the omissions model. As a result, optimal protocols in the generalized omissions model are not necessarily optimal when restricted to runs of the omissions model, as the following theorem shows is the case for simultaneous Byzantine agreement.

Theorem 3.32: No optimal protocol for simultaneous Byzantine agreement in the generalized omissions model is optimal when restricted to runs of the omissions model.

Proof: Let π be the failure pattern (involving at least $2t$ processors) in which processor p_i fails to send to processor p_{t+i} in round 1 (for $i = 1, \ldots, t$) and no other failures occur. Notice that π is a failure pattern of both the omissions model and the generalized omissions model. Let r be a run of a full-information protocol with the failure pattern π. We claim that some nonvalid fact about the initial configuration (in fact, the entire initial configuration) must be common knowledge at $(r, 2)$ in the omissions model; and that no nonvalid fact about the initial configuration is common knowledge at $(r, 2)$ in the generalized omissions model, from which it follows by Corollary 3.3 that no nonvalid fact about the initial configuration is common knowledge at time 2 in any run with failure pattern π of a protocol in the generalized omissions model. In the first case, any optimal protocol for simultaneous Byzantine agreement in the omissions model (the protocol $\mathcal{F}_c$, for example) halts at time 2. In the second case, Lemma 3.1 implies that *no* protocol for simultaneous Byzantine agreement in the generalized omissions model can halt at time 2. Therefore, no optimal protocol for simultaneous Byzantine agreement in the generalized omissions model is optimal when restricted to runs of the omissions model.

To see that some nonvalid fact about the initial configuration becomes common knowledge at $(r, 2)$ in the omissions model, notice that the set $\hat{V}(r, 2)$ is nonempty. The result follows by Corollary 3.18.

To see that no nonvalid fact about the initial configuration becomes common knowledge at $(r, 2)$ in the generalized omissions model, it is enough to show that $(r, 2) \sim (s, 2)$ for all failure-free runs s. Shifting "pebbles," notice that $(r, 2) \sim (r', 2)$ where r' differs from r only in that processor p_1 is nonfaulty in r' and it is processor p_{t+1} that fails to receive the undelivered message from p_1 to p_{t+1} in round 1. Using Lemma 3.29 we can show that $(r', 2) \sim (r'', 2)$ where r'' differs from r' only in that processor p_{t+1} does not fail to receive the message from processor p_1 in round one. Repeating this procedure we can show that $(r'', 2) \sim (u, 2)$ where u is the failure-free run with the input of r''. It is now possible to use Lemma 3.29 to show that $(s, 2) \sim (s', 2)$ for all failure-free runs s and s'. It follows that $(r, 2) \sim (s, 2)$ for all failure-free runs s, and hence that no nonvalid fact about the initial

configuration is common knowledge at $(r, 2)$. □

We remark that, for most simultaneous choice problems, the counterexample given in the proof of Theorem 3.32 can be used to show that no optimal protocol for this problem in the generalized omissions model is optimal when restricted to runs of the omissions model.

The results of this section indicate that the generalized omissions model seems to be a natural failure model that already displays some of the complex behavior of the more malicious models such as the Byzantine failure models. By this we mean that, just as a processor in a Byzantine model may be confused by which of two other processors are actually faulty processors, a processor in the omissions model hearing of a lost message may be confused by whether the sender or the receiver of the lost message is at fault. We believe that this model is therefore a natural candidate for further study as an intermediate model on the way to understanding the mysteries of fault tolerance in truly malicious failure models.

3.6 Conclusions

This chapter applies the theory of knowledge in distributed systems to the design and analysis of fault-tolerant protocols for a large and interesting class of problems. This is a good example of the power of applying reasoning about knowledge to obtain general, unifying results and a high-level perspective on issues in the study of unreliable systems.

Given the effectiveness of a knowledge-based analysis in the case of simultaneous actions (see also [DM86]), it would be interesting to know whether such an analysis can shed similar light on the case of *eventually* coordinated actions. Dolev, Reischuk, and Strong [DRS82] show that the problem of performing eventually coordinated actions in synchronous systems is quite different from that of performing simultaneous actions. For example, they show that while $t + 1$ is a general lower bound on the number of rounds required to reach simultaneous agreement even when the number f of processors actually failing is less that t, eventual agreement can be reached in as few as $f + 2$ rounds if the number of processors is sufficiently large. In addition to common knowledge, an analysis of eventually coordinated actions may be able to make good use of the notion of *eventual common knowledge* (see [HM84, Mos86]). We note that it is possible to show that for eventual

choice problems there do not, in general, exist protocols that are *optimal in all runs*. For example, one can give two protocols for (eventual) Byzantine agreement with the property that for every operating environment one of these protocols will reach Byzantine agreement (i.e., all processors will decide on a value) by time 2 at the latest. However, if $t > 1$, it is well-known that no single protocol can guarantee that agreement will be reached by time 2 in all runs. What is the best notion of optimality that *can* be achieved in eventual coordination?

We provide a method of deriving an optimal protocol for any given *implementable* specification of a simultaneous choice problem. However, in this work, we have completely sidestepped the interesting question of characterizing the problems that are and are not implementable in different failure models. We believe that a general analysis of the implementability of problems involving coordinated actions in different failure models will expose many of the important operational differences between the models. As an example, our specification of the distributed firing squad problem in the introduction is implementable in the variants of the omissions model, but *is not* implementable in more malevolent models, in which a faulty processor can falsely claim to have received a "start" message and otherwise seem to behave correctly (see [BL87] and [CDDS85] for definitions of versions of the firing squad problem that *are* implementable in the more malicious models).

In the generalized omissions model, we have shown how to derive optimal protocols for nontrivial simultaneous choice problems, requiring processors to perform polynomial-space computations between consecutive rounds. We have also shown an NP-hard lower bound for any communication-efficient protocol for such a problem that is optimal in all runs. Determining the precise complexity of this task is a nontrivial open problem, due to the interesting combinatorial structure underlying the generalized omissions model. It would also be interesting to extend our study to more malicious failure models, such as the *Byzantine* and the *authenticated Byzantine* models (see [Fis83]). It is not immediately clear whether the notion of a failure pattern can be defined in these models in a protocol-independent fashion. Thus, it is not clear that the notion of optimality in all runs is well-defined in such models.[7] If such definitions are possible, we believe that the NP-hardness

[7]Quite recently, Michel [Mic89] has shown in the Byzantine model how to map runs of one protocol to runs of another protocol in a way that respects processor failures, and

result from the generalized omissions model should extend to these models. (Our proof does show that testing for common knowledge in runs of a full-information protocol $\mathcal{F}$ in both models is NP-hard.) Capturing the precise combinatorial structure of the similarity graph in these models is bound to expose many of the mysterious properties of the models. We believe that this is an important first step in understanding these models.

As we have seen, there are no computationally-efficient optimal protocols for simultaneous choice problems in the generalized omissions model. Since it is unreasonable to expect polynomial-time processors to perform NP-hard computations, it is natural to ask what is the earliest time at which simultaneous actions can be performed by such processors? Are optimal protocols for such processors guaranteed to exist? In what sense are these protocols optimal? How can they be derived? In contrast to the simpler failure models, the answers to these questions in the generalized omissions model no longer seems to be as closely related to the information-theoretic definitions of knowledge and common knowledge given in Chapter 2, since they do not account for the polynomial-time limitations on processors' computational resources.

A major challenge motivated by these questions, therefore, is the elaboration of the theory of knowledge given in Chapter 2 to include notions of resource-bounded knowledge that would provide us with appropriate tools for analyzing such questions. Such a theory would provide notions such as *polynomial-time knowledge* and *polynomial-time common knowledge*, which would correspond to the actions and the simultaneous actions that polynomial-time processors can perform. Note that the fact that (suboptimal) polynomial-time protocols for the simultaneous Byzantine agreement problem exist even in the more malicious failure models implies that, given the right notions, many relevant facts should become polynomial-time common knowledge.

Recently, in [Mos88], Moses has risen to this challenge and proposed notions of resource-bounded knowledge based on the existence of tests for knowledge similar to the tests for common knowledge used here. Loosely speaking, for example, a processor is said to polynomial-time know a fact φ at a point if it knows φ at this point and there exists a polynomial-time test that at all points of the system correctly determines whether the processor knows φ. Using this definition of polynomial-time knowledge, he shows

how to define a notion of optimality with respect to these mappings.

that polynomial-time common knowledge of certain facts is a necessary condition for processors to perform simultaneous actions, and, using this and the construction in the proof of Theorem 3.26, he is able to prove that there can be no polynomial-time protocol for simultaneous Byzantine agreement in the generalized omissions model that is optimal in all runs with respect to polynomial-time protocols. We note that other related notions of resource-bounded knowledge have appeared in [HMT88] and [FZ88]. While each of these definitions is well-motivated in each of these works, however, understanding which of these definitions is in general the "correct" definition is still an open problem. We will return to this topic again in Chapter 5 where we study cryptographic protocols in terms of a form of resource-bounded knowledge.

Chapter 4

Knowledge, Probability, and Adversaries

In this chapter, we explore the relationship between knowledge and probability in probabilistic systems.

4.1 Introduction

In a number of areas of research, including distributed computing, artificial intelligence, and economics, we are faced with the problem of understanding a system of agents (possibly processors in a distributed network or consumers in an economic model) that interact in some way. Often, probability plays a role in this interaction: in the context of game theory, for example, an agent might toss a coin in order to determine its next move in a game. As we try to understand these probabilistic systems, we often find ourselves reasoning, at least informally, about knowledge and probability and their interaction. Consider, for example, a probabilistic primality-testing algorithm. Such an algorithm might guarantee that if the input n is a composite number, then with high probability the algorithm will find a "witness" that can be used to verify that n is composite. Loosely speaking, we reason, if an agent runs this algorithm on input n and the algorithm fails to find such a witness, then the

This chapter is joint work with Joe Halpern. A preliminary version of this work will appear in *Proceedings of the 8th Annual ACM Symposium on Principles of Distributed Computing*, August, 1989.

agent knows that n is almost certainly prime, since the agent is guaranteed that the algorithm would almost certainly have found a witness had n been composite.

A number of recent papers have tried to formalize this sort of reasoning about knowledge and probability. Fagin and Halpern [FH88] present an abstract model for knowledge and probability in which they assign to each agent and state a probability space to be used when computing the probability, according to that agent at that state, that a formula φ is true. In their framework, the problem of modeling knowledge and probability reduces to choosing this assignment of probability spaces. Although they show that more than one choice may be reasonable, they do not tell us how to make this choice. One particular (and quite natural) choice is made in [FZ88] and some arguments are presented for its appropriateness; another is made in [HMT88] (Chapter 5) and used to analyze interactive proof systems. It is not initially clear, however, which choice is most appropriate.

In this chapter, we clarify the issues involved in choosing the right assignment of probability spaces. We argue that no single assignment is appropriate in all contexts. Different assignments can be viewed as most appropriate in the context of betting against different adversaries. Thinking in terms of such betting games, a statement such as "I know event E will happen with probability at least α" is meaningless until the powers of our opponent in the betting game have been specified. A strategy that will win a game with probability .99 against a weak adversary may win the game with probability only .33 against a strong adversary. Consequently, even if we are told that a certain strategy will win the game with probability .99 against a certain adversary, we cannot tell whether it is a good strategy until we know the powers of this certain adversary.

We find, however, that the notion of an opponent in a betting game does not fully capture all the subtleties that arise when modeling knowledge and probability in distributed systems. We present a framework with three different types of adversaries, each playing a fundamentally different role. We briefly describe these roles here, and explore them in greater depth in the rest of the chapter.

When we analyze probabilistic protocols, we typically do so in terms of probability distributions on the *runs* or *executions* of the protocol. When we say a protocol is correct with probability .99, we mean the protocol will do the right thing in .99 of the runs. A closer analysis of the situation reveals some

subtleties. In fact, we do *not* have a probability distribution on the entire set of runs. In probabilistic algorithms for testing primality such as those of Rabin [Rab80] and Solovay and Strassen [SS77], for example, we typically do not assume a distribution on the inputs (the numbers to be tested). The only source of probability comes from the coins tossed during the execution of the algorithm. This means that for every fixed input, there is a probability space on the runs of the protocol on that input, rather than there being one probability space on the set of all runs. We can view the choice of input as a nondeterministic choice to which we do not assign a probability. Thus, we prove the algorithm works correctly with high probability for each initial nondeterministic choice. A similar situation arises in probabilistic protocols that are designed to work in the presence of a nondeterministic (perhaps adversarial) scheduler (e.g., [Rab82]). Again, we do not wish to assume some probability of playing a given scheduler. Instead, we factor out the choice of scheduler and prove that the protocol is correct with high probability for each scheduler.

This, then, is the role played by the first type of adversary: to factor out the nondeterminism in the system, allowing us to place a well-defined probability on the set of runs for each fixed adversary. We remark that this need to factor out the nondeterminism is implicit in most analyses of probabilistic protocols, and appears explicitly in [Rab82, Var85, FZ88].

The probability on the runs can be viewed as giving us an *a priori* probability of an event, before the protocol is run. However, the probability an agent places on runs will in general change over time, as a function of information received by the agent in the course of the execution of the protocol. New subtleties arise in analyzing this probability.

Consider a situation with three agents p_1, p_2, and p_3. Agent p_2 tosses a fair coin at time 1 and observes the outcome at time 2, but agents p_1 and p_3 never learn the outcome. What is the probability according to p_1 that the coin lands heads? Clearly at time 1 it should be 1/2. What about at time 2? There is one argument that says the answer should be 1/2. After all, agent p_1 does not learn any more about the coin as a result having tossed it, so why should its probability change? Another argument says that after the coin has been tossed, it does not make sense to say that the probability of heads is 1/2. The coin has either landed heads or it hasn't, so the probability of the coin landing heads is either 0 or 1 (although agent p_1 does not know which). This point of view appears in a number of papers in the philosophical literature

(for example, [vF80, Lew80]). Interestingly, the same issue arises in quantum mechanics, in Schrödinger's famous cat-in-the-box thought experiment (see [Pag82] for a discussion).

We claim that these two choices of probability are best explained in terms of betting games. At time 1, agent p_1 should certainly be willing to accept an offer from either p_2 or p_3 to bet \$1 for a payoff of \$2 if the coin lands heads (assuming p_1 is risk neutral).[1] Half the time the coin will land heads and p_1 will be \$1 ahead, and half the time the coin will land tails and p_1 will lose \$1, but on average p_1 will come out even. On the other hand, agent 1 is clearly not willing to accept such an offer from p_2 at time 2 (since p_2 would presumably offer the bet only when it is sure it will win), although it is still willing to accept this bet from p_3. The point here is that in a betting game, not only is your knowledge important, but also the knowledge of the opponent offering the bet. Betting games are not played in isolation!

Thus, the role played by the second type of adversary in our framework is to model the knowledge of the opponent offering a bet to an agent at a given point in the run. One obvious choice is to assume you are playing against someone whose knowledge is identical to your own. This is what decision theorists implicitly do when talking about an agent's *posterior* probabilities [BG54]; it is also how we can understand the choice of probability space made in [FZ88]. By way of contrast, the choice in [HMT88] corresponds to playing someone who has complete knowledge about the past and knows the outcome of the coin toss; this corresponds to the viewpoint that says that when the coin has landed, the probability of heads is either 0 or 1 (although you may not know which).

A further complication arises when analyzing asynchronous systems. In this case there is a precise sense in which the agent does not even know exactly when the event to which it would like to assign a probability is being tested. Thus we need to consider a third type of adversary in asynchronous systems, whose role is to choose the time. To illustrate the need for this third type of adversary, we give an example of an asynchronous system where there are a number of plausible answers to the question "What is the probability the most recent coin toss landed heads?". It turns out that the different answers correspond to different adversaries choosing the times to perform the test

[1]Informally, an agent is said to be risk neutral if it is willing to accept all bets where its expectation of winning is nonnegative.

in different ways. We remark that the case of asynchronous systems is also considered in [FZ88]. We can understand the assignment of "confidence" made there as corresponding to playing against a certain class of adversaries of this third type.

Having shown that different definitions of probabilistic knowledge correspond to different classes of adversaries, we show, given a class of adversaries, how to construct a definition most appropriate for this class. We formalize our intuition concerning the probability an agent assigns to an event in terms of a betting game between the agent and an opponent. We show that our "most appropriate" definition has the property that it enables an agent to break even in this game, and any other definition with this property must correspond to an opponent even *more* powerful than the actual opponent. These results form the technical core of this chapter.

The rest of the chapter is organized as follows. In the next section, Section 4.2, we consider the problem of putting a probability on the runs of a system; this is where we need the first type of adversary, to factor out the nondeterministic choices. In Section 4.3 we start to consider the issue of how probability should change over time. In Section 4.4 we consider the choices that must be made in a general definition of probabilistic knowledge. In Section 4.5 we consider particular choices of probability assignments that seem reasonable in synchronous systems. Here we consider the second type of adversary, representing the knowledge of the opponent in the betting game. In Section 4.6, we consider asynchronous systems, where we also have to consider the third type of adversary. In Section 4.7 we apply our ideas to analyzing the coordinated attack problem, showing how different notions of probability correspond to different levels of guarantees in coordinated attack. The chapter ends with two appendices. In Appendix 4.A we give the proofs of the results claimed in the chapter, and in Appendix 4.B we discuss some interesting secondary observations related to the rest of the chapter.

4.2 Probability on runs

In order to discuss the probability of events in a distributed system, we must specify a probability space. In this section we show that in order to place a reasonable probability distribution on the runs of a system, it is necessary to postulate the existence of the first type of adversary sketched

in the introduction.

Consider the simple system consisting of a single agent who tosses a fair coin once and halts. This system consists of two runs, one in which the coin comes up heads and one in which the coin comes up tails. The coin toss induces a very natural distribution on the two runs: each is assigned probability 1/2.

Now consider the system (suggested by Moshe Vardi; a variant appears in [FZ88]) consisting of two agents, p_1 and p_2, where p_1 has an input bit and two coins, one fair coin landing heads with probability 1/2 and one biased coin landing heads with probability 2/3. If the input bit is 0, p_1 tosses the fair coin once and halts. If the input bit is 1, p_1 tosses the biased coin and halts. This system consists of four runs of the form (b, c), where b is the value of the input bit and c is the outcome of the coin toss. What is the appropriate probability distribution on the runs of this system? For example, what is the probability of heads?

Clearly the *conditional* probability of heads given that the input bit is 0 should be 1/2, while the conditional probability of heads given the input bit is 1 should be 2/3. But what is the *unconditional* probability of heads? If we are given a distribution on the inputs, then it is easy to answer this question. If we assume, for example, that 0 and 1 are equally likely as input values, then we can compute that the probability of heads is $\frac{1}{2} \cdot \frac{1}{2} + \frac{1}{2} \cdot \frac{2}{3} = \frac{7}{12}$. If we are not given a distribution on the inputs, then the question has no obvious answer. It is tempting to assume, therefore, that such a distribution exists. Often, however, assuming a particular fixed distribution on inputs leads to results about a system that are simply too weak to be of any use. Knowing an algorithm produces the correct answer in .99 of its runs when all inputs are equally likely is of no use when the algorithm is used in the context of a different distribution on the inputs.

To overcome this problem, one might be willing to assume the existence of some fixed but unknown distribution on the inputs. Proving that an algorithm produces the correct answer in .99 of the runs in the context of an unknown distribution, however, is no easier than proving that for each fixed input the algorithm is correct in .99 of the runs, since it is always possible for the unknown distribution to place all the probability on the input for which the algorithm performs particularly poorly. Here the advantage of viewing the system as a single probability space is lost, since this is precisely the proof technique one would use when no distribution is assumed in the first

place. Moreover, assuming the existence of some unknown distribution on the inputs simply moves all problems arising from nondeterminism up one level. Although we have a distribution on the space of input values, we have no distribution on the space of probability distributions.

This discussion leads us to conclude that some choices in a distributed system must be viewed as inherently nondeterministic (or, perhaps better, nonprobabilistic), and that it is inappropriate, both philosophically and pragmatically, to model probabilistically what is inherently nondeterministic. But then how can we reason probabilistically about a system involving both nondeterministic and probabilistic choices? Our solution—which is essentially a formalization of the standard approach taken in the literature—is to factor out initial nondeterministic events, and view the system as a collection of subsystems, each with its natural probability distribution. In the coin tossing example above, we would consider two probability spaces, one corresponding to the input bit being 0 and the other corresponding to the input bit being 1. The probability of heads is 1/2 in the first space and 2/3 in the second.[2]

We want to stress that although this example may seem artificial, analogous examples frequently arise in the literature. In a probabilistic primality-testing algorithm [Rab80, SS77], for example, we do not want to assume

[2]Often, even in the presence of nondeterminism, we *can* impose a meaningful distribution on the runs of a system without factoring the system into subsystems. However, the resulting distribution still may not capture all of our intuition. The problem in the preceding example is that probabilistic events (the coin toss) depend on nonprobabilistic events (the input bit). Suppose, however, the agent tosses a fair coin regardless of the input bit's value. Now it is natural to assign probability 1/2 to each of the events $\{(1,h),(0,h)\}$ and $\{(1,t),(0,t)\}$ that the coin lands head and tails, respectively. Consider, however, the situation (discussed in [FH88, HMT88]) where an agent performs a given action a iff the input bit is 1 and the coin landed heads, or the input bit is 0 and the coin landed tails. It is natural to argue that the probability the agent performs the action a is also 1/2: if the input bit is 1 then with probability 1/2 the coin will land heads and a will be performed; and if the input bit is 0 then with probability 1/2 the coin will land tails and a will be performed. Unfortunately, our "natural" distribution on the runs of the system does not support this line of reasoning, since this distribution does not assign a probability to the set $\{(1,h),(0,t)\}$ corresponding to the performance of a. In fact, it is not hard to see that if we could assign this set a probability, then we would be able to assign a probability to having the input bit set to 0 or 1. But the setting of the input bit was assumed to be nondeterministic! Again, however, if we factor out this initial nondeterminism, we can view the system as two subsystems with obvious associated probability distributions, and within each subsystem the action a is performed with probability 1/2. And this is precisely what the reasoning underlying our intuition is implicitly doing.

a probability distribution on the inputs. We want to know that for each choice of input, the algorithm gives the right answer with high probability. Rabin's primality-testing algorithm [Rab80] is based on the existence of a polynomial-time computable predicate $P_n(a)$ with the following properties: (1) if n is composite, then at least 3/4 of the $a \in \{1, \ldots, n-1\}$ cause $P_n(a)$ to be true, and (2) if n is prime, then no such a causes n to be true. Rabin's algorithm generates a polynomial number of a's at random. If $P_n(a)$ is true for any of the a's generated, then the algorithm outputs "composite"; otherwise it outputs "prime". Property (2) guarantees that if the algorithm outputs "composite", then n is definitely composite. If the algorithm outputs "prime", then there is a chance that n is not prime, but property (1) guarantees that this is very rarely the case: if n is indeed composite, then with high probability the algorithm outputs "composite". If the algorithm outputs "prime", therefore, it might seem natural to say that n is prime with high probability; but, of course, this is not quite right. The input n is either prime or it is not; it does not make sense to say that it is prime with high probability. On the other hand, it does make sense to say that the algorithm gives the correct answer with high probability. The natural way to make this statement precise is to partition the runs of the algorithm into a collection of subsystems, one for each possible input, and prove that the algorithm gives the right answer with high probability in each of these subsystems, where the probability on the runs in each subsystem is generated by the random choices for a. While for a fixed composite input n there may be a few runs where the algorithm incorrectly outputs "prime", in almost all runs it will give the correct output.

In many contexts of interest, the choice of input is not the only source of nondeterminism in the system. Later nondeterministic choices may also be made throughout a run. In asynchronous distributed systems, for example, it is common to view the choice of the next processor to take a step or the next message to be delivered as a nondeterministic choice. Similar arguments to those made above can be used to show that we need to factor out these nondeterministic choices in order to use the probabilistic choices (coin tosses) to place a well-defined probability on the set of runs. A common technique for factoring out these nondeterministic choices is to assume the existence of a scheduler deterministically choosing (as a function of the history of the system up to that point) the next processor to take a step (cf. [Rab82, Var85]). It is standard practice to fix some class of schedulers, perhaps the

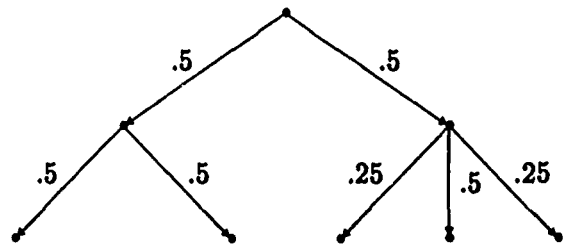


Figure 4.1: A (labeled) computation tree.

class of "fair" schedulers or "polynomial-time" schedulers, and argue that for every scheduler in this class the system satisfies some condition.

As we now show, if we view all nondeterministic choices as under the control of some adversary from some class of adversaries, then there is a straightforward way to view the set of runs of a system as a collection of probability spaces, one for each adversary. By fixing an adversary we factor out the nondeterministic choices and are left with a purely probabilistic system, with the obvious distribution on the runs determined by the probabilistic choices made during the runs. This is essentially the approach taken in [FZ88].

Once we fix an adversary A, we can view the runs of the system with this adversary as a (labeled) *computation tree* $\mathcal{T}_A$ (see Figure 4.1). Nodes of the tree are global states and paths in the tree are runs. Now, however, edges of the tree are labeled with positive real numbers such that for every node the values labeling the node's outgoing edges sum to 1. Intuitively, the value labeling an outgoing edge of node s represents the probability the system makes the corresponding transition from node s.[3] Given a finite path in the

[3]Since all edges have positive labels, we are effectively ignoring transitions with probability 0, and assuming that there is a discrete probability distribution on the set of possible transitions at each node. It follows that each node can have at most a countable number of outgoing edges. This means, for example, that we are disallowing the possibility that the next step will be a random assignment to a variable x chosen with uniform probability from the interval $[0, 1]$. We could easily extend our model to deal with this situation by assigning probabilities to sets of transitions, rather than just individual transitions. We have chosen to consider only discrete probability distributions here for ease of exposition.

tree, the probability of the set of runs extending this finite path is simply the product of the probabilities labeling the edges in this finite path.

It is natural to view this computation tree $\mathcal{T}_A$ as a probability space, a tuple $(\mathcal{R}_A, \mathcal{X}_A, \mu_A)$ where $\mathcal{R}_A$ is the set of runs in $\mathcal{T}_A$, $\mathcal{X}_A$ consists of subsets of $\mathcal{R}_A$ that are *measurable* (that is, the ones to which a probability can be assigned; these are generated by starting with sets of runs with a common finite prefix and closing under countable union and complementation), and a probability function μ_A defined on sets in $\mathcal{X}_A$ so that the probability of a set of runs with a common prefix is the product of the probabilities labeling the edges of the prefix. If we restrict attention to finite runs (as is done in [FZ88]), then it is easy to see that each individual run is measurable, so that $\mathcal{X}_A$ consists of all possible subsets of $\mathcal{R}_A$. Moreover, in the case of finite runs, the probability of a run is just the product of the transition probabilities along the edges of the run.

It is occasionally useful to view this computation tree $\mathcal{T}_A$ as consisting of two components: the tree structure (that is, the unlabeled graph itself), and the assignment of transition probabilities to the edges of the tree. Given an unlabeled tree $\mathcal{T}_A$, we define a *transition probability assignment* for $\mathcal{T}_A$ to be a mapping τ assigning transition probabilities to the edges of $\mathcal{T}_A$. We will use the notation $\mathcal{T}_A$ at times to refer to the unlabeled tree, to the labeled tree, and to the induced probability space; which is meant should be clear from context.

We define a *probabilistic system* to consist of a collection of labeled computation trees (which we view as separate probability spaces), one for each adversary A in some set $\mathcal{A}$. We assume that the environment component in each global state in $\mathcal{T}_A$ encodes the adversary A and the entire past history of the run. This technical assumption ensures that different nodes in the same computation tree have different global states, and that we cannot have the same global state in two different computation trees. Given a point c, we denote the computation tree containing c by $\mathcal{T}(c)$. Our technical assumption guarantees that $\mathcal{T}(c)$ is well-defined.

The choice of the appropriate set $\mathcal{A}$ of adversaries against which the system runs is typically made by the system designer when specifying correctness conditions for the system. An adversary might be limited to choosing the initial input of the agents (in which case the set of possible adversaries would correspond to the set of possible inputs) as is the case in the context of primality-testing algorithms in which an agent receives a single number

(the number to be tested) as input. On the other hand, an adversary may also determine the order in which agents are allowed to take steps, the order in which messages arrive, or the order in which processors fail. One might also wish to restrict the computational power of the adversary to polynomial time. It depends on the application.

4.3 Probability at a point

We are interested in understanding knowledge and probability in distributed systems. An agent's knowledge varies over time, as its state changes. We would expect the probability an agent assigns to an event to vary over time as well. Clearly an agent's probability distribution at a given point must somehow be related to the distribution on runs if it is to be at all meaningful. Nevertheless, the two distributions (the overall distribution on the runs of a system and the distribution on the runs an agent uses at a point) are quite different; depending on which of the distributions we use, we can be led to quite different analyses of a protocol.

To understand this distinction, consider the Coordinated Attack problem [Gra78]. Two generals A and B must decide whether to attack a common enemy, but we require that any attack be a coordinated attack; that is, A attacks iff B attacks. Unfortunately, they can communicate only by messengers who may be captured by the enemy. It is known that it is impossible for the generals to coordinate an attack under such conditions [Gra78, HM84]. Suppose, however, we relax this condition and require only that the generals coordinate their attack with high probability [FH88, FZ88]. To eliminate all nondeterminism, let us assume general A tosses a fair coin to determine whether to attack, and let us assume the probability a messenger is lost to the enemy is 1/2. Our new correctness condition is that the condition "A attacks iff B attacks" holds with probability .99.

Consider the following two-step solution CA_1 to the problem. At round 0, A tosses a coin and sends 10 messengers to B iff the coin landed heads. At round 1, B sends a messenger to tell A whether it has learned the outcome of the coin toss. At round 2, A attacks iff the coin landed heads (regardless of what it hears from B) and B attacks iff at round 1 it learned that the coin landed heads. It is not hard to see that if we put the natural probability space on the set of runs, then with probability at least .99 (taken over the

runs) A attacks iff B attacks: if the coin lands tails then neither attacks, and if the coin lands heads then with probability at least .99 at least one of the ten messengers sent from A to B at round 0 avoids capture and both generals attack.

This is very different, however, from saying that at all times both generals know that with probability at least .99 the attack will be coordinated. To see this, consider the state just before attacking in which A has decided to attack but has received a message from B saying that B has not learned the outcome of the coin toss. At this point, A is certain the attack will not be coordinated. Although we have not yet given a formal definition of how to compute an agent's probability at a given point, it seems unreasonable for an agent to believe with high probability that an event will occur when information available to the agent guarantees it will not occur.

On the other hand, consider the solution CA_2 differing from the preceding one only in that B does not try to send a messenger to A at round 1 informing A about whether B has learned the outcome of the coin toss. An easy argument shows that in this protocol, at all times both generals have confidence (in some sense of the word) at least .99 that the attack will be coordinated. Consider B, for example, after having failed to receive a message from A. B reasons that either A's coin landed tails and neither general will attack, which would happen with probability 1/2, or A's coin landed heads and all messengers were lost, which would happen with probability $1/2^{11}$; and hence the conditional probability that the attack will be coordinated given that B received no messages from A is at least .99.

As the preceding discussion shows, in a protocol which has a certain property P with high probability taken over the runs, an agent may still find itself in a state where it knows perfectly well that P does not (and will not) hold. While correctness conditions P for problems arising in computer science have typically been stated in terms of a probability distribution on the runs, it might be of interest to consider protocols where an agent knows P with high probability at all points. As we shall show, the probability distribution on the runs typically corresponds to each agent's probability distribution at time 0. Thus, we can view the probability on the runs as an *a priori* probability distribution. To require a fact (or a condition P) to hold with high probability from each agent's point of view at all times is typically a much stronger requirement than requiring it to hold with high probability over the set of runs. Arguably, in many cases, it is also a more natural

requirement. It seems quite natural, for example, to require of a coordinated attack protocol that A have high confidence at all points that the attack will be coordinated, rather than allowing A to attack even when it is certain the attack will be uncoordinated.

4.4 Definitions of probabilistic knowledge

We want to make sense of statements such as "at the point c, agent p_i knows φ holds with probability α". The problem is that, although we typically have a well-defined probability distribution on the set of runs in each computation tree, in order to make sense of such statements we need a probability distribution on the points p_i considers possible at c. The reason we need a distribution on points and not just on runs is that many interesting facts are facts about points and not about runs. Consider, for example, the fact "the most recent coin tossed landed heads". If a coin is tossed many times in a single run, this fact may be true at some points of the run and false at others, and hence is a fact about points and not about runs. When reasoning about probabilistic protocols, it seems quite natural to want to make formal statements of the form "agent p knows with probability 1/2 that the most recent coin tossed by agent q landed heads". It is possible to reformulate this statement so that it becomes a fact about runs. The fact "the kth coin tossed by agent q landed heads" is a fact about runs; and the statement above can be reformulated as "for all times k, if the current time is k, then agent p knows with probability 1/2 that the kth coin tossed by agent q landed heads". In our opinion, the former statement more naturally corresponds to the way we think about such protocols. If we are willing to restrict our attention to facts about the run, then we can make do simply with a distribution on runs, but this precludes (or at least complicates) the discussion of many interesting events in a system.

We begin by reviewing the general framework of [FH88] in which, given a particular assignment of probability spaces to points and agents, we can make sense of such statements about an agent's probabilistic knowledge. The remainder of the chapter will focus on the construction of appropriate probability assignments.

Define a *probability assignment* $\mathcal{P}$ to be a mapping from an agent p_i and point c to a probability space $\mathcal{P}_{i,c} = (S_{i,c}, \mathcal{X}_{i,c}, \mu_{i,c})$. Here $S_{i,c}$ is a set of

points, $\mathcal{X}_{i,c}$ is the set of measurable subsets of $S_{i,c}$, and $\mu_{i,c}$ is a probability function assigning a probability to the sets in $\mathcal{X}_{i,c}$.[4] In most cases of interest, one can think of $S_{i,c}$ as a subset of the points agent p_i considers possible at c, and of $\mu_{i,c}$ as indicating the relative likelihood according to p_i that a particular point in $S_{i,c}$ is actually the current point c.[5]

Given such an assignment, let $S_{i,c}(\varphi)$ be the set of the points in $S_{i,c}$ satisfying φ; that is, $S_{i,c}(\varphi) = \{d \in S_{i,c} : d \models \varphi\}$. It is natural to interpret $\mu_{i,c}(S_{i,c}(\varphi))$ as the probability φ is true, according to agent p_i at the point c. One problem with this interpretation, of course, is that the set $S_{i,c}(\varphi)$ is not guaranteed to be measurable, and hence $\mu_{i,c}(S_{i,c}(\varphi))$ is not guaranteed to be well-defined. In order to deal with this problem, we follow the approach of [FH88], and make use of inner and outer measures. Given a probability space $(S, \mathcal{X}, \mu)$, the *inner measure* μ_* and *outer measure* μ^* are defined by

$$\mu_*(S') = \sup\{\mu(T) : T \subseteq S' \textit{ and } T \in \mathcal{X}\}$$
$$\mu^*(S') = \inf\{\mu(T) : T \supseteq S' \textit{ and } T \in \mathcal{X}\}$$

for all subsets S' of S. Roughly speaking, the inner (resp. outer) measure of $S_{i,c}(\varphi)$ is the best lower (resp. upper) bound on the probability φ is true, according to p_i at c. It is easy to see that $\mu^*(T) = 1 - \mu_*(T^c)$ for any set T, where T^c is the complement of T. Given a probability assignment $\mathcal{P}$, we write $\mathcal{P}, c \models Pr_i(\varphi) \geq \alpha$ to mean $\mu_{i,c_*}(S_{i,c}(\varphi)) \geq \alpha$.[6] Note that we need the probability assignment $\mathcal{P}$ to make sense of Pr_i. We take $K_i^\alpha\varphi$ to be an abbreviation for $K_i(Pr_i(\varphi) \geq \alpha)$; thus $K_i^\alpha\varphi$ means that agent p_i knows that the probability of φ is at least α since $Pr_i(\varphi) \geq \alpha$ holds at all points p_i considers possible.

We now have all the definitions needed to give semantics to a logical language of knowledge and probability. In particular, the language of most

[4] We often follow the standard practice [Hal50, p. 73] of identifying the probability space $\mathcal{P}_{i,c}$ with the sample space $S_{i,c}$; the intention should be clear from context.

[5] Returning to the question of distributions on runs versus points, notice that as long as the set $S_{i,c}$ does not contain more than one point per run, there is a natural bijection from the probability on the points in $S_{i,c}$ to the probability on the runs going through $S_{i,c}$. In general, however, we allow more than one point on the same run to appear in $S_{i,c}$. As we shall see in the next section, this generality is useful when dealing with asynchronous systems.

[6] We remark that we can easily extend these definitions to more complicated formulas such as $Pr_i(\varphi) \geq 2Pr_i(\psi)$; see [FH88].

interest to us in the remainder of this chapter is the language $\mathcal{L}(\Phi)$ obtained by fixing a set Φ of primitive propositions and closing under the standard boolean connectives (conjunction and negation), the knowledge operators K_i, probability formulas of the form $Pr_i(\varphi) \geq \alpha$, and the standard (linear time) temporal logic operators next $\bigcirc$ and until U. Note that $\mathcal{L}(\Phi)$ is sufficiently powerful to express the operators K_i^α and the temporal operators henceforth $\Box$ and eventually $\Diamond$.[7] In the context of a given system, we say that $\mathcal{L}(\Phi)$ is *state-generated* if each of the primitive propositions in Φ is a fact about the global state; and we say that $\mathcal{L}(\Phi)$ is *sufficiently rich* if for every global state g there is a primitive proposition in Φ true at precisely those points with global state g. This condition ensures that the language $\mathcal{L}(\Phi)$ is rich enough to allow us to talk about individual global states. The assumption that $\mathcal{L}(\Phi)$ is state-generated is quite reasonable in practice: we typically take the primitive propositions to represent facts such as "the coin landed heads", "the message was received", or "the value of variable x is 0". Each of these facts is a fact about the global state, assuming certain aspects of the history are recorded in the global state. Sufficient richness is a technical condition required for a few of our results. We can always make a language sufficiently rich by adding primitive propositions.

We now have a natural way of making sense of knowledge and probability, given a probability assignment $\mathcal{P}$. Unfortunately, we still do not know how to choose $\mathcal{P}$, but our choices are somewhat more constrained than they may at first appear. We are given the computation trees and the associated distributions on runs, and we clearly want the distribution on the sample space $S_{i,c}$ of points we associate with agent p_i at point c to be related somehow to these distributions on runs. We next show that once we choose the sample spaces $S_{i,c}$, there is a straightforward way to use the distribution on runs to induce a distribution on $S_{i,c}$. Thus, once we are given an appropriate choice of sample spaces and the distributions on runs of the computation trees, we can construct the probability assignment. The problem of choosing a probability

[7]We define $(r,k) \models \bigcirc\varphi$ iff $(r,k+1) \models \varphi$, so $\bigcirc\varphi$ is true at time k in a run iff it is true at time $k+1$, after the next step. We define $(r,k) \models \varphi\, U\, \psi$ to mean there exists $\ell \geq k$ such that $(r,\ell) \models \psi$ and $(r,\ell') \models \varphi$ for all ℓ' with $k \leq \ell' < \ell$. Thus $\varphi\, U\, \psi$ is true at (r,k) if ψ is true at some point in the future, and φ is true until then. Recall that $\Diamond\varphi$, which says that φ is true at some point in the future, can be taken as an abbreviation of $true\, U\, \varphi$; and that $\Box\varphi$, which says that φ is true now and forever in the future, is an abbreviation for $\neg\Diamond\neg\varphi$.

assignment, therefore, essentially reduces to choosing the sample spaces. This reduction will clarify important issues in determining the appropriate choice of probability assignments.

The idea of our construction is quite straightforward: given a sample space $S_{i,c}$ and a subset $S \subseteq S_{i,c}$, the probability of S (relative to $S_{i,c}$) is just the probability of the runs going through S normalized by the probability of the set of runs going through $S_{i,c}$. In other words, the probability of S is the conditional probability a run passes through S, given that the run passes through $S_{i,c}$.

In order for this simple idea to work, however, the set $S_{i,c}$ must satisfy a few requirements. One natural choice for $S_{i,c}$ is the set $\mathcal{K}_i(c)$ of all points agent p_i considers possible at c. In general, however, this set contains points from many different computation trees, and attempting to impose a distribution on this set of points leads to the same difficulties that led us to factor out nondeterminism and view a system as a collection of computation trees in the first place. Recall the example from Section 3 in which p_1 tosses a fair or biased coin, depending on whether its input is 0 or 1. Before (and after) the coin is tossed, p_2 considers four worlds possible, one from each possible run. We can no more place a probability on these points than we could place a probability on the four runs. On the other hand, given a point c from a run with input bit 1 (corresponding to the biased coin), if we restrict $S_{2,c}$ to consist of the two points in the computation tree with input 1, then we can put a probability on the two points in the obvious way and compute the probability of heads as 2/3. This intuition leads us to require that each set $S_{i,c}$ be contained entirely within a single computation tree:

REQ_1. *All points of $S_{i,c}$ are in $\mathcal{T}(c)$.*

We remark that, while REQ_1 does not allow us to take $S_{i,c}$ to be all of $\mathcal{K}_i(c)$, it still seems natural to choose $S_{i,c} \subseteq \mathcal{K}_i(c)$. We say that a probability assignment is *consistent* if it satisfies this condition. As pointed out in [FH88], a consequence of this is that if p_i knows φ, then φ holds with probability 1; that is, $K_i(\varphi) \Rightarrow (Pr_i(\varphi) = 1)$.[8] With a consistent assignment, it cannot be the case that agent p_i both knows φ and at the same time assigns $\neg\varphi$ positive probability.

[8]In fact, as pointed out in [FH88], this axiom characterises the property that the probability space used by p_i is a subset of the points that p_i considers possible.

The single condition REQ_1, however, is not enough for our idea for imposing a distribution on the set $S_{i,c}$ of points to work. Because this idea involves conditioning on the set of runs passing through $S_{i,c}$, the definition of conditional probability forces us to require that that this set of runs is a measurable set with positive measure. Suppose $\mathcal{T}(c) = (\mathcal{R}_A, \mathcal{X}_A, \mu_A)$, for some adversary A. Given a set S of points contained in $\mathcal{T}(c)$, denote by $\mathcal{R}(S)$ the set of runs passing through S; that is, $\mathcal{R}(S) = \{r \in \mathcal{R}_A : (r,k) \in S \text{ for some } k\}$. We require that

$$REQ_2\text{: } \mathcal{R}(S_{i,c}) \in \mathcal{X}_A \textit{ and } \mu_A(\mathcal{R}(S_{i,c})) > 0.$$

REQ_2 is a relatively weak requirement. The following lemma shows that, in practice, REQ_2 is typically satisfied. A set S of points is said to be *state-generated* if $(r,k) \in S$ and $r(k) = r'(k')$ imply $(r',k') \in S$; in other words, S contains all points with the same global state as (r,k).

Proposition 4.1: If $S_{i,c}$ is state-generated and satisfies REQ_1, then $S_{i,c}$ satisfies REQ_2.

The proof of Proposition 4.1 (and all other technical results in this chapter) can be found in Appendix 4.A. We remark that this statement is actually independent of the transition probability assignment τ assigning probabilities to the edges of T_A. While REQ_2 seems to depend on both $S_{i,c}$ and τ, Proposition 4.1 tells us we can choose $S_{i,c}$ without regard for τ and be confident REQ_2 will be satisfied for whatever τ we eventually choose, as long as $S_{i,c}$ is state-generated.

Given a set of points $S_{i,c}$ satisfying REQ_1 and REQ_2, we now make precise our idea for imposing a distribution on $S_{i,c}$. Intuitively, to construct the collection $\mathcal{X}_{i,c}$ of measurable subsets of $S_{i,c}$, we project the measurable subsets of the runs of $\mathcal{T}(c)$ onto $S_{i,c}$. Formally, given a set $\mathcal{R}'$ of runs and a set S of points, we define $Proj(\mathcal{R}', S) = \{(r,k) \in S : r \in \mathcal{R}'\}$. We define

$$\mathcal{X}_{i,c} = \{Proj(\mathcal{R}', S_{i,c}) : \mathcal{R}' \in \mathcal{X}_A\}.$$

Finally, we define the probability function $\mu_{i,c}$ on the measurable subsets of $S_{i,c}$ via conditional probability:

$$\mu_{i,c}(S) = \mu_A(\mathcal{R}(S) \mid \mathcal{R}(S_{i,c})) = \frac{\mu_A(\mathcal{R}(S))}{\mu_A(\mathcal{R}(S_{i,c}))}$$

for all $S \in \mathcal{X}_{i,c}$. Let $P_{i,c} = (S_{i,c}, \mathcal{X}_{i,c}, \mu_{i,c})$.

Proposition 4.2: If $S_{i,c}$ satisfies REQ_1 and REQ_2, then $P_{i,c}$ is a probability space.

We can now formalize our intuition that the construction of probability assignments reduces to the choice of sample spaces. Given a system (i.e. a collection of labeled computation trees), define a ***sample space assignment*** to be a function $\mathcal{S}$ that assigns to each agent p_i and point c a sample space $\mathcal{S}(i,c) = S_{i,c}$ satisfying REQ_1 and REQ_2. Given a sample space assignment $\mathcal{S}$, our construction shows how to obtain a probability space $\mathcal{P}_{i,c}$ for all agents p_i and all points c. This naturally determines a probability assignment $\mathcal{P}$, which we call the the ***probability assignment induced by*** $\mathcal{S}$. We note that the definition of $\mathcal{P}$ actually depends on ***both*** the sample space assignment $\mathcal{S}$ and the transition probability assignment τ (implicitly determined by the fact that we have labeled computation trees). There are times when it is convenient to start with an unlabeled computation tree, labeled by some transition probability assignment τ. In this case, we refer to $\mathcal{P}$ as the ***probability assignment induced by*** $\mathcal{S}$ ***and*** τ. For future reference, we define a fact φ to be ***measurable with respect to*** $\mathcal{S}$ if $S_{i,c}(\varphi) \in \mathcal{X}_{i,c}$ for all agents p_i and points c.

The preceding discussion makes precise the idea that choosing a probability assignment reduces to choosing a sample space assignment, but still does not help us choose the sample space assignment. Different choices result in probability assignments with quite different properties. Let us return to the example in the introduction, where p_1 tosses a fair coin, and neither p_2 nor p_3 observe the outcome. Clearly, at time 2 (after the coin has been tossed), p_2 considers two points possible: say h (the coin landed heads) and t (the coin landed tails). Consider the sample space assignment $\mathcal{S}^1$ such that $\mathcal{S}^1(2,h) = \mathcal{S}^1(2,t) = \{h,t\}$. Thus, at both of the points h and t, the same sample space is being used. In this case, at both points, the probability of heads is 1/2. Thus, with respect to the induced probability assignment, p_2 knows that the probability of heads is 1/2. On the other hand, consider assignment $\mathcal{S}^2$ such that $\mathcal{S}^2(2,h) = \{h\}$ and $\mathcal{S}^2(2,t) = \{t\}$. With respect to the induced probability assignment, the probability of heads at h according to p_2 is 1, while the probability of heads at t is 0. In this case, all that p_2 can say is that it knows that the probability of heads is either 1 or 0, but it doesn't know which. Which is the right probability assignment? As we hinted in the introduction, the answer depends on another type of adversary, the one that p_2 views itself as playing against. This is the focal point of the

next section.

We conclude this section with one further example. Consider a system where a fair die is tossed by p_1 and p_2 does not know the outcome. Suppose that at time 2 the die has already been tossed. Let $c_1, \ldots c_6$ be the six points corresponding to the possible outcomes of the die. What sample space assignment should we use for p_2? One obvious choice is to take the assignment $\mathcal{S}^1$ which assigns the same sample space at all six points, the space consisting of all the points. With respect to this sample space, each point will have probability 1/6. Let φ be the statement "the die landed on an even number". Clearly, in the probability space induced by this sample space, φ holds with probability 1/2. Since p_2 uses the same sample space at all six points, agent p_2 knows that the probability of φ is 1/2. A second possibility is to consider two sample spaces $S_1 = \{c_1, c_2, c_3\}$ and $S_2 = \{c_4, c_5, c_6\}$; let the assignment $\mathcal{S}^2$ assign the sample space S_1 to agent p_2 at all the points in S_1, and the sample space S_2 at all the points in S_2. Thus, at all the points in S_1, the probability of φ is 1/3, while at all the points in S_2, the probability of φ is 2/3. All p_2 can say is that it knows that the probability of φ is either 1/3 or 2/3, but it does not know which.

Clearly we can subdivide the six points into even smaller subspaces. It is not too hard to show that the more we subdivide, the less precise is p_2's knowledge of the probability. (We prove a formal version of this statement in the next section.) But why bother subdividing? Why not stick to the first sample space assignment, which gives the most precise (and seemingly natural) answer? Our reply is that, again, this may not be the appropriate answer when playing against certain adversaries.

4.5 Probability in synchronous systems

We first consider the problem of selecting appropriate probability assignments in completely synchronous systems. Intuitively, a system is synchronous if all agents effectively have access to a global clock. Recall from Chapter 2 that a system is *synchronous* [HV89] if for all points (r, k) and (r', k') and all agents p_i, if $r_i(k) = r'_i(k')$ then $k = k'$. Again, this means that no two points an agent p_i considers indistinguishable can lie on the same run.

When considering probability, it turns out that many things become much easier in the context of synchronous systems. For example, it turns out

that, in practice, sample space assignments satisfy three natural properties: (a) they are state-generated; (b) they are *inclusive*, which means $c \in S_{i,c}$ for all agents p_i and points c; and (c) they are *uniform*, which means that $d \in S_{i,c}$ implies $S_{i,d} = S_{i,c}$ for all agents p_i and points c and d.[9] We say that $\mathcal{S}$ (and its induced probability assignment) is *standard* if it satisfies these three properties. For the remainder of this section we consider only standard assignments.

One convenient feature of synchronous systems is that all facts of interest are measurable. Recall that $\mathcal{L}(\Phi)$ is state-generated with respect to a system $\mathcal{R}$ if all the primitive propositions in Φ are facts about the global state.

Proposition 4.3: In a synchronous system, if $\mathcal{S}$ is a consistent standard assignment and $\mathcal{L}(\Phi)$ is state-generated, then φ is measurable with respect to $\mathcal{S}$ for all facts $\varphi \in \mathcal{L}(\Phi)$.

This result says that for all practical purposes we do not have to concern ourselves with nonmeasurable sets and inner measures in synchronous systems. The proof is by induction on the structure of φ, and can be found in Appendix 4.A.

We begin our examination of probability assignments in synchronous systems by defining four sample space assignments and their induced probability assignments. Each of these assignments can be understood in terms of a betting game against an appropriate opponent. (This is the second type of adversary mentioned in the introduction.) We make this intuition precise after we have defined the probability assignments.

The first of these assignments corresponds to what decision theorists would call an agent's *posterior* probability. This is essentially the probability an agent would assign to an event given everything the agent knows. This intuitively corresponds to the bet an agent would be willing to accept from a copy of itself, someone with precisely the same knowledge that it has. We make this relationship between probability and betting precise shortly.

What probability space corresponds to an agent's conditioning on its knowledge in this way? Since we have identified an agent p_i's knowledge with the set of points p_i considers possible at c, this set of points seems the most

[9] Condition (c) is essentially the definition of a uniform probability assignment from [FH88]. A probability assignment induced by a uniform sample space assignment as we have defined it here is a uniform probability assignment in the sense of [FH88].

natural choice for the space. As we have seen, however, this set of points is not in general contained in one computation tree. Thus, we consider instead the set of points in c's computation tree $\mathcal{T}(c)$ that p_i considers possible at c. This is just the set $Tree_{i,c} = \{d \in \mathcal{T}(c) : c \sim_i d\}$. It is clear that $Tree_{i,c}$ satisfies REQ_1; that it satisfies REQ_2 follows by Proposition 4.1 since it is state-generated. By Proposition 4.2, therefore, the induced probability space $(Tree_{i,c}, \mathcal{X}_{i,c}, \mu_{i,c})$ is indeed a probability space. Let $\mathcal{S}^{post}$ be the sample space assignment that assigns the space $Tree_{i,c}$ to agent p_i at the point c, and let $\mathcal{P}^{post}$ be the probability assignment induced by $\mathcal{S}^{post}$.

The probability space $\mathcal{P}^{post}_{i,c}$ has a natural interpretation. It is generated by conditioning on everything p_i knows at the point c and the fact that it is playing against the adversary A that generated the tree $\mathcal{T}_A$ in which c lies. Of course, the agent considers many adversaries possible. Thus, the statement $\mathcal{P}^{post}, c \models K_i^{\alpha}\varphi$ means that for all adversaries p_i considers possible at c (given its information at c), the probability of φ given all p_i knows is at least α. $\mathcal{P}^{post}$ is precisely the assignment advocated in [FZ88] in the synchronous case.

Suppose now that p_i were considering accepting a bet from someone (not necessarily an agent in the system) with complete knowledge of the past history of the system. In this case, we claim that the appropriate choice of probability space for p_i at the point $c = (r, k)$ is all the other points (r', k) that have the same prefix as (r, k) up to time k; in other words, all points with the global state $r(k)$. Call this set of points $Pref_{i,c}$. Note that $Pref_{i,c}$ is independent of p_i, and depends only on the point c. Moreover, $Pref_{i,c}$ is clearly state-generated (by $r(k)$ itself), so by Propositions 4.1 and 4.2 we can again induce a natural probability distribution on this set of points by conditioning on the runs passing through $Pref_{i,c}$. Let $\mathcal{S}^{fut}$ denote the sample space assignment that assigns $Pref_{i,c}$ to p_i at c, and let $\mathcal{P}^{fut}$ denote the probability assignment induced by $\mathcal{S}^{fut}$. We remark that this is the probability assignment used in [HMT88], as well as [LS82].

In the probability space $\mathcal{P}^{fut}_{i,c}$, any event that has already happened by the point c will have probability 1. Future events (that get decided further down the computation tree) still have nontrivial probabilities, which is why we have termed it a future probability assignment.

Let us reconsider yet again the coin tossing example from the introduction, where agent p_2 tosses a fair coin at time 1 but agents p_1 and p_3 do not learn the outcome. Since the coin has already landed at time 2, it is easy to check that we have $\mathcal{P}^{fut}, c \models K_1(Pr_1(heads) = 1 \vee Pr_1(heads) = 0)$. On the

other hand, we have $\mathcal{P}^{post}, c \models K_1(Pr_1(heads) = 1/2)$. Thus, $\mathcal{P}^{post}$ and $\mathcal{P}^{fut}$ correspond to the two natural answers we considered for the probability of heads. They capture the intuition that the answer depends on the knowledge of the opponent p_1 is betting against: $\mathcal{P}^{fut}$ corresponds to betting against p_2, and $\mathcal{P}^{post}$ corresponds to betting against p_3.

Notice that in both the cases of $\mathcal{P}^{post}$ and $\mathcal{P}^{fut}$, the probability space associated with an agent at a point corresponds to the set of points the agent and its opponent *both* consider possible. Suppose, in general, that p_i is considering what an appropriate bet to accept from p_j would be. We claim (and show below) that in this case the probability assignment should be generated by the *joint* knowledge of agents p_i and p_j, as represented by the intersection of the points they both consider possible; that is, by the set $Tree^j_{i,c} = Tree_{i,c} \cap Tree_{j,c}$. (Note that $Tree^i_{i,c} = Tree_{i,c}$, so that this construction can be viewed as a generalization of the previous one.) Again it is easy to see that $Tree^j_{i,c}$ is state-generated, so by Propositions 4.1 and 4.2 we can induce the natural distribution on this set of points by conditioning on the runs passing through $Tree^j_{i,c}$. Let $\mathcal{S}^j$ be the sample space assignment that assigns $Tree^j_{i,c}$ to p_i at c, and let $\mathcal{P}^j$ be the probability assignment induced by $\mathcal{S}^j$.

All the examples we have seen up to now—$\mathcal{S}^{post}$, $\mathcal{S}^{fut}$, $\mathcal{S}^j$, and $\mathcal{S}^{prior}$—have had the property that $S_{i,c} \subseteq \mathcal{K}_i(c)$, which means they are consistent. As mentioned in Section 4.4, such assignments are characterized by the intuitively desirable condition $K_i(\varphi) \Rightarrow (Pr_i(\varphi) = 1)$; when we return to the coordinated attack problem in Section 4.7, we will see an example of an inconsistent assignment which causes an agent to know the attack will be coordinated with high probability, while knowing that the attack will not be coordinated(!). While consistency seems a natural restriction on probability assignments, it is not a requirement of our framework. There may be be technical reasons for considering inconsistent assignments. One obvious (although inconsistent) probability assignment associates with the point (r, k) the set of *all* time k points in its computation tree. Call this set $All_{i,c}$. ($All_{i,c}$ is in fact independent of p_i.) The probability space induced by the construction of Proposition 4.2 in this case simulates the probability on the runs. Let us denote the associated sample space and probability assignments by $\mathcal{S}^{prior}$ and $\mathcal{P}^{prior}$. Notice that if p_i uses the probability space $\mathcal{P}^{prior}_{i,c}$, it is essentially ignoring all that it has learned up to the point c, which is why we have termed it a prior probability.

All four of the sample space assignments we have constructed are standard assignments. It is not difficult to see, in fact, that any assignment constructed on the basis of some opponent's knowledge will be standard. This lends some justification to our restriction to standard assignments. We can view these four assignments as points in a lattice of all possible standard sample space assignments. We define an ordering $\leq$ on this lattice by $\mathcal{S}' \leq \mathcal{S}$ iff $S'_{i,c} \subseteq S_{i,c}$ for every agent p_i and point c. An important property of this ordering is the following:

Proposition 4.4: If $\mathcal{S}$ and $\mathcal{S}'$ are standard assignments satisfying $\mathcal{S}' \leq \mathcal{S}$, then for every agent p_i and point c, the set $S_{i,c}$ can be partitioned into sets of the form $S'_{i,d}$ with $d \in S_{i,c}$.

Intuitively, this means that the sets $S'_{i,c}$ are refinements of the sets $S_{i,c}$, since the sets $S'_{i,c}$ are obtained by carving the sets $S_{i,c}$ into pieces. Consider $\mathcal{S}^{post}$ and $\mathcal{S}^{fut}$, for example. Every set $Tree_{i,c}$ of $\mathcal{S}^{post}$ can be partitioned into the sets $Tree^j_{i,d}$ of $\mathcal{S}^{fut}$ with $d \in Tree_{i,c}$. In fact, it is clear that

$$\mathcal{S}^{fut} \leq \mathcal{S}^j \leq \mathcal{S}^{post} \leq \mathcal{S}^{prior}.$$

Furthermore, notice that $\mathcal{S}^{post}$ is greatest (with respect to $\leq$) among all consistent sample space assignments.

In the case of consistent assignments, if we interpret $S_{i,c}$ as the intersection of p_i's knowledge with its opponent's knowledge, we can think of $\mathcal{S}' \leq \mathcal{S}$ as roughly meaning that the opponent corresponding to $\mathcal{S}'$ considers fewer points possible and hence knows more than the opponent corresponding to $\mathcal{S}$. This means, for example, that $\mathcal{S}^{post}$, as the maximal consistent assignment, corresponds to playing against the least powerful opponent.

The ordering on sample spaces assignments induces an obvious ordering on probability assignments: given two sample space assignments $\mathcal{S}'$ and $\mathcal{S}$ and their induced probability assignments $\mathcal{P}'$ and $\mathcal{P}$, respectively, we define $\mathcal{P}' \leq \mathcal{P}$ iff $\mathcal{S}' \leq \mathcal{S}$. An important point to note is that if $\mathcal{P}'$ and $\mathcal{P}$ are consistent assignments satisfying $\mathcal{P}' \leq \mathcal{P}$, then $\mu'_{i,c}$ can be obtained from $\mu_{i,c}$ by conditioning with respect to $S'_{i,c}$:

Proposition 4.5: In a synchronous system, if $\mathcal{P}'$ and $\mathcal{P}$ are consistent standard assignments satisfying $\mathcal{P}' \leq \mathcal{P}$, then for all agents p_i, all points c, and all measurable subsets $S' \in \mathcal{X}'_{i,c}$

(a) $S' \in \mathcal{X}_{i,c}$ (so that, in particular, $S'_{i,c}$ itself is a measurable subset of $S_{i,c}$),

(b) $\mu_{i,c}(S'_{i,c}) > 0$,

(c) $\mu'_{i,c}(S') = \mu_{i,c}(S'|S'_{i,c}) = \frac{\mu_{i,c}(S')}{\mu_{i,c}(S'_{i,c})}$.

It follows that any consistent probability assignment can be obtained from $\mathcal{P}^{post}$ by conditioning.

We are now able to make precise the sense in which $\mathcal{P}^{post}$, $\mathcal{P}^j$, and $\mathcal{P}^{fut}$ are the "right" probability assignments for an agent to use when playing against an opponent who knows exactly as much as it does, when playing against p_j, and when playing against an opponent who has complete information about the past. We focus on $\mathcal{P}^j$ here, but the arguments are the same in all cases.

Consider the following betting game between agents p_i and p_j at a point c. Agent p_j offers p_i a payoff of β for a bet on φ. Agent p_i either accepts or rejects the bet. If p_i accepts the bet, p_i pays one dollar to p_j in order to play the game, and p_j pays β dollars to p_i if φ is true at c. Thus, if p_i accepts this bet at the point c, then p_i's net gain is either $\beta - 1$ or -1 depending on whether φ is true or false at c; if p_i rejects the bet, we say its gain is 0.

Intuitively, assuming that p_i is risk neutral, p_i can always be convinced to accept a bet on φ no matter how low the probability of φ is, as long as p_i believes there is some nontrivial chance φ is true and the payoff β is high enough. Our intuition says there must be some relationship between the probability α with which p_i knows φ and this acceptable payoff β that would induce p_i to accept a bet on φ. If α is close to 0 then p_i might require a high payoff to make the bet's risk acceptable, while if α is close to 1 then p_i might be willing to accept a much lower payoff since the chance of losing is so remote. Our claim that $\mathcal{P}^j$ is the right probability assignment is based on the fact that $\mathcal{P}^j$ determines for an agent p_i the lowest acceptable payoff for a bet with p_j on a fact φ. In other words, $\mathcal{P}^j$ determines precisely how an agent p_i should bet when betting against p_j. In fact, $\mathcal{P}^j$ is in a sense the unique such probability assignment. We now make this intuition precise.

What should p_i consider an acceptable payoff for a bet on φ, assuming p_i does not want to lose money on the bet? Since p_j is presumably following some strategy for offering bets to p_i, the acceptable payoff should take this strategy into account. Consider, for example, the system in which p_j secretly

tosses a fair coin at time 0, and offers at time 1 to bet p_i that the coin landed heads. If p_j is following the strategy of always offering a payoff of \$2, independent of the outcome of the coin toss, then p_i can always safely accept the bet since, on average, it will not lose any money (that is, p_i's expected profit is zero). If p_j offers a payoff of \$2 only when the coin lands tails, then p_i is certain to lose money. On the other hand, if p_j offers a payoff of \$2 only when the coin lands heads, then it is p_j who is certain to lose money. While we expect that p_j will not follow a strategy that will cause it to lose money, we assume only that p_j's strategy for offering bets depends only on its local state. In other words, given two points p_j is unable to distinguish, p_j must offer the same payoff for a bet on φ at both points. Formally, a *strategy* for p_j is a function from p_j's local state at a point c to the payoff p_j should offer p_i for a bet on φ at c. Similarly, we assume that p_i's strategy for accepting or rejecting bets (that is, for computing acceptable payoffs) is also a function of its local state.

Again, what should p_i consider an acceptable payoff for a bet on φ? Suppose p_i decides it will accept any bet on φ with a payoff of at least $1/\alpha$ when its local state is s_i (remember that p_i's strategy for accepting bets must be a function of its local state). Denoting by $Bet(\varphi, \alpha)$ the rule "accept any bet on φ with a payoff of at least $1/\alpha$", how well does p_i do by following $Bet(\varphi, \alpha)$ when its local state is s_i? Clearly p_i will win some bets and lose others, so we are interested in computing p_i's expected profit. This in turn depends on p_j's strategy. This leads us to compute, for each of p_j's strategies f, agent p_i's expected profit when p_i follows $Bet(\varphi, \alpha)$ and p_j follows f. Intuitively, if, for each of p_j's strategies f, agent p_i's expected profit is nonnegative, then p_i does not lose money on average by following $Bet(\varphi, \alpha)$, regardless of p_j's strategy.

Before we can compute p_i's expected profit, however, there is an important question to answer: What probability space should we use to compute this expectation at a point c? One reasonable choice is to take $Tree_{i,c}$; this would correspond to computing this expectation with respect to everything p_i knows. Another reasonable choice would be to take $Tree^j_{i,c}$. The intuition would be that p_i wants to do well for every possible choice of what p_j could do to p_i. The sets $Tree^j_{i,c}$ correspond to the different things p_j could do, since p_j's strategy is a function of its local state. For definiteness, we take the expectation with respect to the probability space $Tree^j_{i,c}$ here, and then show that our results would not have been affected (at least in the synchronous

setting) if we had chosen the space $Tree_{i,c}$ instead.

Let the value of the random variable $W_f = W_f(\varphi, \alpha)$ at a point d denote p_i's profit (or winnings) at d, assuming p_i is following $Bet(\varphi, \alpha)$ and p_j is following f. Assume that φ is measurable with respect to $\mathcal{S}^j$. Let $E_{i,c}[W_f] = E_{Tree^j_{i,c}}[W_f]$ denote the expected value of W_f with respect to the probability space $Tree^j_{i,c}$. We say p_i *breaks even* with $Bet(\varphi, \alpha)$ at c if $E_{i,c}[W_f] \geq 0$ for every strategy f for p_j. We say the rule $Bet(\varphi, \alpha)$ is *safe* for p_i at c if p_i breaks even with $Bet(\varphi, \alpha)$ at all points p_i considers possible at c.

To justify our definition of safe bets, we now prove that the definition remains unchanged if we take the expectation with respect to $Tree_{i,c}$ instead of $Tree^j_{i,c}$. We define $\text{Tree}^j_{i,c}$-*safe* to mean safe as defined above, and $\text{Tree}_{i,c}$-*safe* just as we defined safe, except that now we take the expectation with respect to $Tree_{i,c}$ instead of $Tree^j_{i,c}$.

Proposition 4.6: In a synchronous system, for all facts φ, all agents p_i, and all points c, the rule $Bet(\varphi, \alpha)$ is $Tree_{i,c}$-safe for p_i at c iff $Bet(\varphi, \alpha)$ is $Tree^j_{i,c}$-safe for p_i at c.

Our claim that $\mathcal{P}^j$ is the right probability assignment to use when playing against p_j is made concrete by the following result which states that $\mathcal{P}^j$ determines for every agent p_i precisely what bets are safe when betting against p_j.

Theorem 4.7: For all facts φ measurable with respect to $\mathcal{P}^j$, all agents p_i, and all points c, the rule $Bet(\varphi, \alpha)$ is safe for p_i at c iff $\mathcal{P}^j, c \models K_i^\alpha \varphi$.

We view this as the main result of this chapter. It says that that $\mathcal{P}^j$ determines precisely what bets are safe for p_i to accept. If, using the probability assignment $\mathcal{P}^j$, agent p_i knows the probability of φ is at least α, then p_i will at least break even betting on φ when the payoff is $1/\alpha$. On the other hand, if, using $\mathcal{P}^j$, agent p_i considers it possible that the probability of φ is less than α, then there is a strategy p_j can use that causes p_i to lose money betting on φ when the payoff is $1/\alpha$. In other words, $\mathcal{P}^j$ is the right probability assignment to use when betting against p_j.

While this theorem is stated only for *measurable* facts φ, remember that Proposition 4.3 assures us that facts of interest are typically measurable in synchronous systems. In fact, the same theorem holds even for nonmeasurable facts, once we define an appropriate notion of expectation for such facts; we consider this notion in Appendix 4.B.2

The proof of Theorem 4.7 depends only on the fact that $\mathcal{P}^j$ is induced by $\mathcal{S}^j$, and is actually independent of the particular transition probability assignment τ determining the distribution on runs. In this sense it is really $\mathcal{S}^j$ that is determining what bets are safe for p_i to accept. We can formalize this intuition as follows. We say that a standard sample space assignment $\mathcal{S}$ *determines safe bets* against p_j in a system consisting of unlabeled computation trees if, for all transition probability assignments τ assigning transition probabilities to edges of the computation trees, the following condition holds for the probability assignment $\mathcal{P}$ induced by $\mathcal{S}$ and τ:

$$\mathcal{P}, c \models K_i^\alpha \varphi \text{ implies } Bet(\varphi, \alpha) \text{ is safe for } p_i \text{ at } c$$

for all facts $\varphi \in \mathcal{L}(\Phi)$, all agents p_i, and all points c. Notice that this definition quantifies over all transition probability assignments τ, requiring that the probability assignment induced by $\mathcal{S}$ determines safe bets regardless of the actual choice of τ. Our intuition says that the "right" way to go about constructing a probability assignment should not depend on the details of the transition probabilities. We would like some uniform way of choosing the probability space that does not change if there are small perturbations in the probability; Theorem 4.7 shows us that it is always possible to construct an assignment $\mathcal{P}^j$ in this way.

While the proof of Theorem 4.7 shows that $\mathcal{S}^j$ determines safe bets against p_j, it turns out that there are other assignments that determine safe bets against p_j. If the language $\mathcal{L}(\Phi)$ is sufficiently rich, however, so that there are a lot of possible events that can be bet on, then $\mathcal{S}^j$ enjoys the distinction of being the maximum such assignment.

Theorem 4.8: In a synchronous system, if $\mathcal{S}$ is a consistent standard assignment, then

(a) if $\mathcal{S} \leq \mathcal{S}^j$, then $\mathcal{S}$ determines safe bets against p_j, and

(b) if $\mathcal{S}$ determines safe bets against p_j and $\mathcal{L}(\Phi)$ is sufficiently rich, then $\mathcal{S} \leq \mathcal{S}^j$.

We interpret Theorems 4.7 and 4.8 as providing strong evidence that $\mathcal{S}^j$ is the right sample space assignment, and hence that $\mathcal{P}^j$ is the right probability assignment, to use when playing against an opponent with p_j's knowledge. It says that the only way for p_i to be guaranteed it is using a safe betting

strategy against p_j is by assuming the opponent is at least as powerful as p_j. Intuitively, the more powerful the opponent the less confident the agent is that it will be able to win a bet with this opponent, and the higher the payoff the agent will require before accepting a bet. Consequently, p_i is being unduly conservative if it takes a probability assignment that corresponds to an agent that is more powerful than p_j since it may pass up bets it should accept.[10]

In the process of making this intuition precise, we can prove a theorem that gives us further insight into relationships between sample space assignments on the lattice. Recall that we have defined $K_i^\alpha\varphi$ to mean agent p_i knows α is a *lower* bound on the probability of φ. We can extend this definition to deal with intervals in a straightforward way. We would like to define $K_i^{[\alpha,\beta]}\varphi$ to mean $K_i(\alpha \leq Pr_i(\varphi) \leq \beta)$, which should mean agent p_i knows the probability of φ is somewhere between α and β. Since φ may not correspond to a measurable set, what we really mean is that the inner measure of φ is at least α and the outer measure is at most β. Since we interpret Pr_i as inner measure when φ does not correspond to a measurable set, and since $\mu^*(T) = 1 - \mu_*(T^c)$ for any set T, we can capture this intuition in terms of our language by interpreting $K_i^{[\alpha,\beta]}\varphi$ as an abbreviation for $K_i[(Pr_i(\varphi)) \geq \alpha) \wedge (Pr_i(\neg\varphi) \geq 1-\beta)]$. To relate this definition to our earlier definition of $K_i^\alpha\varphi$, notice that $K_i^\alpha\varphi$ is equivalent to $K_i^{[\alpha,1]}\varphi$. We can now prove the following.

Theorem 4.9: In a synchronous system, if $\mathcal{P}'$ and $\mathcal{P}$ are consistent standard assignments satisfying $\mathcal{P}' < \mathcal{P}$, then

(a) for every fact φ, every agent p_i, every point c, and all α, β with $0 \leq \alpha \leq \beta \leq 1$, we have

$$\mathcal{P}', c \models K_i^{[\alpha,\beta]}\varphi \text{ implies } \mathcal{P}, c \models K_i^{[\alpha,\beta]}\varphi,$$

[10]Strictly speaking, we should justify the fact that p_i should use a rule of the form $Bet(\varphi, \alpha)$ in order to determine when to accept a bet. After all, why should such a simple threshold function be appropriate? It is conceivable that a better money-making strategy might tell p_i, say, to accept a bet on φ if the offered payoff is in the interval $[2, 5]$ or $[8, 10]$, and reject the bet otherwise. It is not hard to show, however, that because we make no assumption about the strategy being followed by p_j (other than requiring that it be a function of p_j's local state), this second strategy is safe for p_i at c iff it is safe for p_i at c to accept a bet on φ if the offered payoff is in the interval $[2, \infty)$, i.e. if $Bet(\varphi, 1/2)$ is safe for p_i at c. Consequently an optimal strategy may as well be taken to be a threshold function like $Bet(\varphi, \alpha)$.

(b) there exist a fact φ, an agent p_i, a point c, and α, β with $0 \leq \alpha \leq \beta \leq 1$ such that

$$\mathcal{P}', c \not\models K_i^{[\alpha,1]}\varphi \text{ and yet } \mathcal{P}, c \models K_i^{[\alpha,1]}\varphi$$

$$\mathcal{P}', c \not\models K_j^{[0,\beta]}\neg\varphi \text{ and yet } \mathcal{P}, c \models K_j^{[0,\beta]}\neg\varphi.$$

If $\mathcal{L}(\Phi)$ is sufficiently rich, then $\varphi \in \mathcal{L}(\Phi)$.

Part (a) shows that an agent's confidence interval does not increase in the presence of a more powerful opponent; part (b) shows that it might actually decrease. The formula φ from part (b) gives an example of a case that agent p_i might be unduly conservative by using an inappropriate probability assignment: using $\mathcal{P}'$, agent p_i would reject bets on φ with payoff $1/\alpha$ even though it should be accepting all such bets.

Our results show that $\mathcal{P}^{post}$ has a special status among probability assignments. It is a maximum assignment among consistent assignments in the lattice with the $\leq$ ordering, and so, by Theorem 4.9, gives the sharpest bounds on the probability interval among all consistent probability assignments. In addition, any other consistent probability assignment can be obtained from $\mathcal{P}^{post}$ by a process of conditioning. Finally, $\mathcal{P}^{post}$ is the probability assignment that corresponds to what decision theorists seem to use when referring to an agent's subjective (or posterior) probability. However, as we have seen, $\mathcal{P}^{post}$ may not always be the "right" probability assignment to use. The right choice depends on the knowledge of the opponent offering us the bet in the system we wish to analyze. Although $\mathcal{P}^{post}$ may give a smaller interval than $\mathcal{P}^j$ (intuitively giving sharper bounds on an agent's belief a fact is true), if p_i uses the better lower bound from $\mathcal{P}^{post}$ as a guide to deciding what bet to accept from p_j, it may wind up losing money. In fact, it follows from Theorems 4.8 and 4.9 that $\mathcal{P}^j$ is the probability assignment that gives an agent the best interval and still guarantees a good betting strategy.

Even in cases where $\mathcal{P}^{post}$ is the "right" choice, it is not necessarily the probability we want to use in computations. It may not always be necessary to obtain the sharpest interval of confidence possible. A rough bound may be sufficient. Theorem 4.9 shows that proving a lower bound on an agent's confidence using a certain choice of probability space implies the same bound holds with any definition higher in the lattice. The advantage of using a probability assignment that lies lower in lattice is that, because the individual probability spaces are smaller, the computations may be simpler. Consider

the definition $\mathcal{P}^{fut}$, for example. Here the probability space we associate with a point (r, k) consists only of points (r', k) having the same global state as (r, k). The runs r' are the runs extending the global state $r(k)$. This means we can reason about the probability of a future event given a fixed global state. In contrast a definition such as $\mathcal{P}^{post}$ allows for the possibility that the runs r' may extend any of a collection of global states, which may mean we no longer have the luxury of arguing about the probability of a future event given a fixed global state. When arguing about the level of confidence of an agent, it seems best to choose a definition as low in the lattice as possible to make the proof as simple as possible, but high enough to enable one to prove a sufficiently high level of confidence.

4.6 Probability in asynchronous systems

We now turn our attention to choosing appropriate probability assignments in asynchronous systems. We remark that even in the context of asynchronous systems, the four sample space assignments discussed in the previous section—$\mathcal{S}^{post}$, $\mathcal{S}^{fut}$, $\mathcal{S}^{j}$, and $\mathcal{S}^{prior}$—still make perfect sense. The intuition motivating these definitions remains the same; in particular, Theorem 4.7 which says that $\mathcal{S}^{j}$ determines safe bets against p_j still holds.

A number of things *do* change, however. For one thing, Proposition 4.3 no longer holds, so many facts of interest become nonmeasurable. Equally important, Proposition 4.5, which says that probability assignments further down in the lattice can all be obtained by conditioning from probability assignments higher in the lattice, also fails in general. The reason it may fail is that if $\mathcal{S}' \leq \mathcal{S}$, we are no longer guaranteed that $S'_{i,c}$ is a *measurable* subset of $S_{i,c}$. For example, although $\mathcal{P}^{j} \leq \mathcal{P}^{post}$, $Tree^{j}_{i,c}$ need not be a measurable subset of $Tree_{i,c}$. If p_j can distinguish time 1 points from time 2 points but p_i cannot, and if c is a time 1 point, then $Tree^{j}_{i,c}$ consists only of the time 1 points while $Tree_{i,c}$ consists of the time 1 and 2 points; in this case, $Tree^{j}_{i,c}$ is not a measurable subset of $Tree_{i,c}$. All our conditioning arguments used this measurability assumption. Consequently, it is no longer true that all consistent assignments can be obtained by conditioning on $\mathcal{P}^{post}$. For similar reasons, in general asynchronous systems, using $Tree_{i,c}$ and using $Tree^{j}_{i,c}$ in the definition of a safe bet does not necessarily give the same results. (The conditional probability argument used in the proof of Proposition 4.6 depends

on the fact that the sets $Tree^j_{i,c}$ are measurable subsets of $Tree_{i,c}$.) We *can* prove analogues of Propositions 4.5 and 4.6 as well as Theorem 4.9, provided we assume that $\mathcal{S}' \leq \mathcal{S}$ and that $S'_{i,c}$ is a measurable subset of $S_{i,c}$ for all agents p_i and points c.[11] Unfortunately, as we shall see, this measurability requirement does not hold in many cases of interest.

The situation is perhaps best illustrated by an example. Consider a simple asynchronous system in which agent p_1 tosses a fair coin 10 times and halts; agents p_2 and p_3 do nothing and never learn the outcome of the coin tosses. This system consists of a single computation tree, a complete binary tree of depth 10 with every transition labeled 1/2. Suppose agent p_2 does not have access to a clock, and so is unable to distinguish any of the global states in the tree. On the other hand, p_3 does have a clock, and so can tell each time apart.

There are clearly 2^{10} possible runs in the system, one corresponding to each of the possible sequences of coin tosses. Since p_2 cannot distinguish any point on any of these runs, for every point c, the set $S^{post}_{2,c}$ consists of every point in the system. Which subsets of $S^{post}_{2,c}$ are measurable? Since the computation tree is finite, each individual run is a measurable set, so all sets of runs are measurable. And since the measurable subsets of $S^{post}_{2,c}$ are obtained by projecting measurable subsets of runs onto $S^{post}_{2,c}$, the sets in $\mathcal{X}^{post}_{2,c}$ are those consisting of all the points on some set of runs in the computation tree.

Let φ be the fact "the most recent coin toss landed heads". Although this is a fact about the global state, the set of points where it is true is not a measurable subset of $S^{post}_{2,c}$, since it does not consist of all the points on some subset of runs. This already shows that Proposition 4.3 fails in this case. Thus, we cannot talk about the probability that p_2 knows φ at a point c in the tree. We can talk about the inner and outer measure of $S^{post}_{2,c}(\varphi)$, however. Since the only nontrivial measurable set contained in $S_{2,c}(\varphi)$ is the set of points on the single run in which the coin lands heads every time, the inner measure of this set is $1/2^{10}$; similarly, the outer measure is $1-(1/2^{10})$.

While values such as $1/2^{10}$ and $1-(1/2^{10})$ may seem somewhat strange

[11]In part (b) of this analogue of Theorem 4.9, we must also strengthen the definition of sufficiently rich to mean that for every global state there is a primitive proposition in Φ true at all points of all runs passing through this global state. This is due to the fact that consistent assignments in asynchronous systems allow a set $S_{i,c}$ to contain more than one point of a given run.

at first glance, they are not totally unmotivated. Consider the situation of agent p_2 at a point c trying to figure out the probability of heads, given only the probability on the runs. Agent p_2 has no idea which run it is in. The only run in which it is always the case that the most recent coin toss landed heads is the run where the coin lands heads on every toss: this run occurs with probability $1/2^{10}$. On the other hand, in all the runs except for the one in which the coin lands tails on every toss, it is possible that the most recent coin toss landed heads. Thus, in a set of runs of probability $1-(1/2^{10})$, it is possible that the most recent coin toss landed heads. This means that $1/2^{10}$ and $1-1/2^{10}$—the inner and outer measure of $S^{post}_{2,c}(\varphi)$—provide lower and upper bounds on the probability of being in a run where the most recent coin toss landed heads.

Now suppose that agent p_2 is betting against p_3. Since p_3 knows what the time is, the sets $S^3_{2,(r,k)}$ consist of all the time k points. With respect to the sample space assignment $\mathcal{S}^3$, the fact φ is measurable. In fact, it's easy to see that $\mu^3(S_{2,c}(\varphi)) = 1/2$ for all points c. To sum up, we have $\mathcal{P}^{post}, c \models K_2^{[1/2^{10},1-(1/2^{10})]}\varphi$ and $\mathcal{P}^{post}, c \models \neg K_2^{1/2}\varphi$, while $\mathcal{P}^3, c \models K_2^{1/2}\varphi$.[12]

This may seem somewhat counterintuitive, since it seems to suggest that p_2 must play more conservatively against a copy of itself than against p_3, who knows more. This is especially so since there is another line of reasoning about this situation which would lead p_2 to conclude that it knows that the probability that the most recent coin toss landed heads is 1/2, even without considering p_3. Agent p_2 reasons as follows: "The current time is k, although I do not know what k is. Regardless of the particular value of k, the probability that the kth coin toss lands heads is 1/2, and hence I know the most recent coin toss landed heads with probability 1/2." The sample space assignment that captures this intuition would associate with the point (r,k) and agent p_2 the set of *time k points* in (r,k)'s computation tree agent p_2 considers possible at (r,k) (as opposed to considering *all* the points in the computation tree that p_2 considers possible, as is done by $\mathcal{P}^{post}$). But this is precisely the assignment $\mathcal{S}^3$!

In order to understand this situation a little better, let us reconsider the assignment $\mathcal{P}^{post}$. We claim that the reason the interval $[1/2^{10},(1-1/2^{10})]$

[12]Note that this does not contradict Theorem 4.9, since Theorem 4.9 would hold only if $S^3_{i,c}$ is a measurable subset of $S_{i,c}$ for all p_i and c, which we have already noted is not the case.

arises here is different from the reason intervals arise in the context of the synchronous systems studied in the preceding section. In the context of synchronous systems, because p_j's strategy depends on its local state and p_i does not know which local state p_j is currently in, p_i has to partition $\mathcal{K}_i(c)$ and view each element of the partition as an independent probability space, computing the probability of φ separately in each one. A formula such as $K_i^{[\alpha,\beta]}\varphi$ holds when the probability of φ can range from α to β in the different probability spaces. In our current example, however, there is only one probability space; the interval arises because of the nonmeasurability of φ. Depending on how "lucky" p_1 is in the choice of where in each run it tests for heads, the probability of getting heads could range from $1/2^{10}$ to $1-(1/2^{10})$.

We can view the nonmeasurability that arises due to asynchrony as a new element of uncertainty that an adversary can exploit. Intuitively, in the coin tossing example, when p_1 plays against (a copy of) itself, since p_1 does not know where in the run it is, an adversary gets to choose that. On the other hand, when playing against p_3, at least p_1 knows that all the worlds in a given sample space are time k points, for some fixed k. We can view our analysis where we obtain the answer 1/2 without invoking p_3 as implicitly assuming an adversary who chooses the time k the test for φ is to be performed. Such an adversary is an adversary of the third type mentioned in the introduction. Given any time k chosen by this adversary, the probability of φ is 1/2.

We can formalize this analysis as follows. With each time k we associate a separate computation tree corresponding to the adversary A_k choosing time k to test for φ. The probability space for p_2 at each point in the tree corresponding to A_k consists of the time k points in the tree, each of which is assigned equal probability. In each of these probability spaces the probability of heads is 1/2, so p_2 knows that the most recent coin toss landed heads with probability 1/2.

There is no reason, however, to restrict this third type of adversary to simply making an initial choice of the stopping time. Suppose we have fixed a collection of adversaries of the first type (the computation trees) and an adversary of the second type (say p_j). We define a *cut* through $Tree^j_{i,c}$ to be a subset of $Tree^j_{i,c}$ containing precisely one point from every run passing through $Tree^j_{i,c}$: every run passing through $Tree^j_{i,c}$ is cut precisely once by such a set of points. We define a type three adversary to be a function

mapping an agent p_i and a point c to a cut through $Tree^j_{i,c}$. Intuitively, p_i and p_j are betting on a fact φ, but neither knows precisely where in the run the bet is taking place; it is the third type of adversary who determines where in the run the bet is actually made. The cut through $Tree^j_{i,c}$ chosen by the adversary is the set of points at which the adversary will cause the bet to take place when the local states of p_i and p_j are given by c.

In the example above, when p_1 plays against a copy of itself, the adversary chooses one cut per computation tree, since p_1 considers all points in the computation tree possible. In the case of p_1 playing against p_3 (who knows the time), the adversary chooses one cut for every time k; this cut must in fact consist of all time k points in the tree. (In general, if we are considering a set of time k points, the only allowable cut is the one consisting of all points. This is why the issue of an adversary choosing such cuts does not arise when considering synchronous systems.)

To make formal sense of this, suppose we are given a set $\mathcal{A}$ of type one adversaries (determining the possible initial nondeterministic choices). This determines a set of computation trees, as we have already discussed. Fix a type two adversary, say p_j. Let $\mathcal{C}$ be a set of type three adversaries in this collection of computation trees (so that the adversaries in $\mathcal{C}$ choose stopping times). Notice that the definition of $\mathcal{C}$ depends on $\mathcal{A}$ and p_j. We can then construct one computation tree $\mathcal{T}_{A,C}$ for each $A \in \mathcal{A}$ and $C \in \mathcal{C}$. For a fixed $A \in \mathcal{A}$, the computation trees $\mathcal{T}_{A,C}$ look identical (essentially just like $\mathcal{T}_A$) for all choices of $C \in \mathcal{C}$ except that we put C into the environment state at each point in $\mathcal{T}_{A,C}$. The sample space assignment $\mathcal{S}^{\mathcal{C}}$ maps an agent p_i and a point c of a tree $\mathcal{T}_{A,C}$ to a sample space $S^{\mathcal{C}}_{i,c} \subseteq Tree^j_{i,c}$ such that for each run $r \in \mathcal{R}(Tree^j_{i,c})$, exactly one point $(r,k) \in Tree^j_{i,c}$ is in $S^{\mathcal{C}}_{i,c}$. Intuitively, this is the point in r where the test is performed. Note that if we consider two adversaries $C, C' \in \mathcal{C}$ and two corresponding points c and c' in $\mathcal{T}_{A,C}$ and $\mathcal{T}_{A,C'}$, the sample spaces $S_{i,c}$ and $S_{i,c'}$ used by p_i at these two points will in general be different: at c, it is C that determines at which point in each run in the tree that p_i considers possible at c the test will be performed, while at c' it is C' that makes this determination. Notice that, in the presence of this third type of adversary, it is no longer the case that all sample space assignments defined in asynchronous systems are standard assignments as they are in synchronous systems. For example, it no longer need be the case that $c \in S^{\mathcal{C}}_{i,c}$.

Intuitively, playing against a copy of yourself places no constraints on this

third type of adversary. To make this precise, once we fix a set of adversaries of the first type $\mathcal{A}$ and consider the resulting system, we can take $pts(\mathcal{A})$ to be the set of all possible adversaries of the third type in this system.

Proposition 4.10: $\mathcal{P}^{post}, c \models K_i^{[\alpha,\beta]}\varphi$ iff $\mathcal{P}^{pts}, c \models K_i^{[\alpha,\beta]}\varphi$, for every fact φ, agent p_i, and point c.

The proof of this result shows that $\mathcal{P}^{post}$ can be understood in asynchronous systems in terms of an adversary that chooses as the time for the test to be performed the worst possible time from p_i's point of view.[13]

Of course, there is no reason to assume that a type three adversary must either be restricted to choosing horizontal cuts of time k points or be allowed to choose completely arbitrary cuts of points. Other intermediate definitions seem plausible as well. One can imagine a partially synchronous model in which processors cannot tell time but are guaranteed that, for every k, all processors take their k^{th} step within some time interval of width δ. It would seem reasonable to require the adversary of the third type, rather than selecting horizontal time k cuts or totally arbitrary cuts, to select cuts with the property that every point in the cut is a time k point for some k falling in some interval of width δ. We can also generalize the notion of type three adversary slightly so as not to require that it choose a cut, but rather have it choose at most one point per run. The intuition here is that this adversary simply does not give p_i the chance to bet in certain runs. In our coin tossing example, such an adversary could allow p_i to bet on heads only when the coin has landed tails. The issue of defining reasonable adversaries of the third type deserves further study.

We close this section with a comparison of our definition of probability in asynchronous systems with that of [FZ88]. The probability assignment used in [FZ88] in the asynchronous setting has much the same flavor as that of our $\mathcal{P}^{pts}$. Rather than assuming that the adversary chooses at a point c a cut of *points* through $Tree_{i,c}$, however, Fischer and Zuck assume that the adversary chooses a cut of *global states* through $Tree_{i,c}$; that is, a set of global states appearing in $Tree_{i,c}$ with the property that no two global states lie on the the same run. Intuitively, this means that if the adversary performs the test at

[13]Another interpretation of this result is that the language obtained by closing a set of formulas under the standard boolean connectives and the modal operators $K_i^{\mathcal{P}}$ cannot distinguish the assignments $\mathcal{P}^{post}$ and $\mathcal{P}^{pts}$. We note that the richer language of [FH88] can distinguish these assignments.

one point, it performs the test at all other points with the same global state. This seems like a reasonable restriction, but it leads to some unexpected consequences.

Let us call the class of adversaries considered in [FZ88] *state*, and let the corresponding probability assignment be $\mathcal{P}^{state}$. Rather than giving formal definitions here, we give an example to show how $\mathcal{P}^{state}$ differs from $\mathcal{P}^{pts}$. Consider a system in which p_1 tosses a biased coin which lands heads with probability .99 and tails with probability .01. The system consists of two runs we can denote by h and t and four points corresponding to times 0 and 1 in runs h and t. The computation tree has only three nodes, a root a, encoding the points $(h,0)$ and $(t,0)$, a node b corresponding to the point $(h,1)$, and a node c corresponding to $(t,1)$. Suppose p_2 is able to distinguish only the point $(h,1)$ from the remaining three points and suppose that φ is the fact "the coin lands heads" (so that φ is true at $(h,0)$ and $(h,1)$, and false elsewhere). Let c be a time 0 point, say $(t,0)$, and consider the probability with which p_i knows φ with respect to $\mathcal{P}^{pts}$ and $\mathcal{P}^{state}$. An adversary in *pts* can either choose $\{(h,0),(t,0)\}$ or $\{(h,0),(t,1)\}$ as the set of points to perform the experiment; φ is true with probability .99 with respect to both sets. It follows that $\mathcal{P}^{pts}, c \models K_2^{.99}\varphi$; in fact we have $\mathcal{P}^{pts}, c \models K_2^{[.99,.99]}\varphi$. Similarly, an adversary in *state* can choose either the node a or the node c as a state at which to perform the experiment, since these are the cuts of global states contained in $\{a,c\}$. The choice of a corresponds to the adversary in *pts* that chooses $\{(h,0),(t,0)\}$. However, the choice of c does not correspond to $\{(h,0),(t,1)\}$. In fact, there is no adversary in *state* corresponding to this adversary in *pts*, since it would amount to choosing the nodes a and c, both of which lie on the same run. With respect to the choice a, φ holds with probability .99; with respect to the choice c, φ holds with probability 0. Thus, we get $\mathcal{P}^{state}, c \models K_2^{[0,.99]}\varphi$. In some sense it seems that $\mathcal{P}^{pts}$ is giving the more reasonable answer here. Since p_2 knows that, *a priori*, the coin will land heads with high probability, and its information has not eliminated either run, it should still consider heads extremely probable.[14]

[14]Note that this example also shows that the adversaries in *state* are examples of the more general adversaries discussed above, that do not necessarily choose one point per run. For example, the adversary choosing the global state c does not choose a point in the run h.

4.7 An application: coordinated attack

As an example of how probabilistic knowledge can be used to analyze protocols, and of how heavily statements made about protocols depend on the particular definition of probabilistic knowledge used, we now apply the different probability assignments defined in the context of synchronous systems to understanding probabilistic coordinated attack as defined in Section 4.3. In [HM84] it is shown that a state of knowledge called *common knowledge* is a necessary condition for coordinated attack. Recall that a formula φ is common knowledge if all agents know φ, all agents know all agents know φ, and so on *ad infinitum*. In the same paper it is shown that common knowledge of nontrivial facts cannot be attained in systems where there is no upper bound on message delivery time (and, in particular, in asynchronous systems), and hence that coordinated attack is not possible in such systems. We now examine the relationship between probabilistic common knowledge and probabilistic coordinated attack.

Recall from Chapter 2 that common knowledge is defined as follows. Given a set $G \subseteq \{p_1, \ldots, p_n\}$ of agents, we define *everyone in G knows* φ by $E_G\varphi \equiv \bigwedge_{p_i \in G} K_i\varphi$. Defining $E_G^k\varphi$ inductively by $E_G^0\varphi = \varphi$ and $E_G^k\varphi = E_G E_G^{k-1}\varphi$, we define φ *is common knowledge to G* by $C_G\varphi \equiv \bigwedge_{k>0} E_G^k\varphi$. Recall that common knowledge satisfies the following statements:

1. *the fixed point axiom*: $C_G\varphi \equiv E_G(\varphi \wedge C_G\varphi)$.

2. *the induction rule*: From $\psi \supset E_G(\psi \wedge \varphi)$ infer $\psi \supset C_G\varphi$.

The first statement says that $C_G\varphi$ is a fixed point of the equation $X \equiv E_G(\varphi \wedge X)$. In fact, it can be shown to follow from the induction rule that $C_G\varphi$ is the *greatest* fixed point, and thus is implied by all other fixed points of this equation [HM85].

By direct analogy, probabilistic common knowledge is defined in [FH88] as the greatest fixed point of the equation $X \equiv E_G^\alpha(\varphi \wedge X)$, where $E_G^\alpha\varphi \equiv \bigwedge_{p_i \in G} K_i^\alpha\varphi$.[15] It is easy to show that the definition of $C_G^\alpha\varphi$ satisfies the obvious analogues of the fixed point axiom and induction rule given above.

[15] As is shown in [FH88], this definition is not equivalent to the infinite conjunction of $(E_G^\alpha)^k\varphi$, $k > 0$; however it is equivalent to the infinite conjunction of $(F_G^\alpha)^k\varphi$, $k > 0$, where we define $(F_G^\alpha)^k\varphi$ inductively by unwinding the fixed point equation: $(F_G^\alpha)^0\varphi = \varphi$ and $(F_G^\alpha)^k\varphi = E_G^\alpha(\varphi \wedge (F_G^\alpha)^{k-1}\varphi)$.

Now consider the probabilistic attack problem, and suppose φ is the fact "A attacks iff B attacks". In the original coordinated attack problem, since φ is true at all points, the induction rule implies $C_G\varphi$ holds at all points. Are there implementations of the probabilistic attack problem where $C_G^\alpha\varphi$ holds at all points? The answer depends on the choice of probability assignment. Stronger assignments yield stronger notions of probabilistic common knowledge which make stronger requirements of the implementation.

Consider the assignment $\mathcal{P}^{fut}$. Here the opponent offering an agent a bet knows the entire global state at every point. If there is any point where the attack is uncoordinated, then no run extending this point can satisfy φ. At this point φ holds with probability 0 (according to $\mathcal{P}^{fut}$), so it easily follows that $C_G^\alpha\varphi$ cannot hold at all points. This says that an algorithm achieves probabilistic coordinated attack with respect to $\mathcal{P}^{fut}$ iff it achieves coordinated attack. Since coordinated attack is known to be unattainable in asynchronous systems, we cannot get probabilistic coordinated attack either with respect to such a strong opponent.

Next consider the assignment $\mathcal{P}^{post}$. Here the opponent offering the bet has precisely the same knowledge as the agent itself. Consequently, if it is possible to reach a point at which the agent can determine from its local state that no run extending the point can satisfy φ, the agent knows φ does not hold, and hence neither does $C_G^\alpha\varphi$. Consequently, our first implementation CA_1 of the probabilistic attack problem does not have the property that $C_G^\alpha\varphi$ holds at all points (with respect to $\mathcal{P}^{post}$), but our second implementation CA_2 does. This can be proved by first observing that $E_G^\alpha\varphi$ holds at all points (with respect to $\mathcal{P}^{post}$) and hence by the induction rule (taking the formula ψ in the rule to be *true*), so does $C_G^\alpha\varphi$.

Notice that with respect to any consistent probability assignment, if at some point an agent in G knows φ does not hold, then $C_G^\alpha\varphi$ cannot hold at this point (since $C_G^\alpha\varphi$ implies $E_G^\alpha\varphi$ by the fixed point axiom, while $K_i\neg\varphi$ implies $\neg E_G^\alpha\varphi$ for all $i \in G$). Consequently, it cannot be the case that $C_G^\alpha\varphi$ holds at all points of CA_1 with respect to any consistent assignment. Is it possible for $C_G^\alpha\varphi$ to hold at all points of CA_1 with respect to *any* probability assignment? Since this algorithm guarantees φ holds with probability α, taken over the runs, the obvious solution is to make the assignment mimic the probability distribution on the runs. In particular, consider $\mathcal{P}^{prior}$. It is easy to see that with this assignment, every agent knows φ with probability α at all points of the system. Since $E_G^\alpha\varphi$ holds at all points, it follows by the

induction rule that $C_G^\alpha\varphi$ holds at all points as well.

We summarize our discussion in the following proposition.

Proposition 4.11:

1. CA_1 achieves probabilistic coordinated attack with respect to $\mathcal{P}^{prior}$ but not $\mathcal{P}^{post}$.

2. CA_2 achieves probabilistic coordinated attack with respect to $\mathcal{P}^{post}$ (and $\mathcal{P}^{prior}$) but not $\mathcal{P}^{fut}$.

3. A protocol achieves probabilistic coordinated attack with respect to $\mathcal{P}^{fut}$ iff it achieves coordinated attack, and hence no such protocol exists in which the generals actually attack.

This proposition shows how increasing the power of the opponent (moving down in the lattice) strengthens the kind of guarantees that can be made for probabilistic attack. Note that all of the probability assignments agree at time 0, and the probability they assign to a set of points is identical to the probability of the set of runs going through those points; i.e. if c is a time 0 point in $\mathcal{T}_A$ and $\mathcal{R}_A(\varphi)$ is the set of runs in $\mathcal{T}_A$ satisfying a fact φ about the run, then

$$\begin{aligned}\mu_A(\mathcal{R}(\varphi)) &= \mu_{i,c}^{post}(Tree_{i,c}(\varphi)) = \mu_{i,c}^{i}(Tree_{i,c}^{i}(\varphi)) \\ &= \mu_{i,c}^{fut}(Pref_{i,c}(\varphi)) = \mu_{i,c}^{prior}(All_{i,c}(\varphi)).\end{aligned}$$

However, at later times, it is only $\mathcal{P}^{prior}$ that agrees with the initial probability on runs. Thus, for the other probability assignments, saying that φ holds with probability greater than α at all points (r,k) in $\mathcal{T}_A$ according to p_i will generally be a stronger statement than saying it holds with probability α taken over the runs of $\mathcal{T}_A$.

Of course, it is perfectly conceivable we might want to consider probability assignments besides those that we have discussed above, which will make yet more guarantees. Considering such intermediate assignments might be particularly appropriate in protocols where security is a major consideration, such as cryptographic protocols. There it becomes quite important to consider the knowledge of the agent we are betting against.

We remark that a slightly different definition of probabilistic coordinated attack is considered in [FZ88]: it is required only that the conditional probability that both parties attack together, given that one of the parties attacks, is at least α.[16] It is then shown in [FZ88] that this form of probabilistic coordinated attack corresponds to all the agents having *average belief* of α that the attack will be coordinated. We can reinterpret these results in our language as showing that this notion of coordinated attack is equivalent to probabilistic common knowledge with respect to another probability assignment, much in the spirit of $\mathcal{P}^{prior}$. In particular, the probability space used by [FZ88] for this analysis is not $\mathcal{P}^{post}$, but an inconsistent probability assignment. However, it should be noted that one can be led to counterintuitive results using an inconsistent probability assignment. Consider $\mathcal{P}^{prior}$ in the context of CA_1. Since there is a point at which the information in agent A's local state guarantees the attack will not be coordinated, according to $\mathcal{P}^{prior}$ both $K_A^\alpha\varphi$ and $K_A\neg\varphi$ hold at this point. In other words, the choice of $\mathcal{P}^{prior}$ has the effect of saying that at a point an agent can have high confidence in a fact it knows to be false.

The preceding discussion raises another interesting point. While it is typically the case that computer science applications consider only probabilities over runs (such applications typically require only that a condition P hold throughout a large fraction of the runs, which corresponds to $\mathcal{P}^{prior}$), it is not clear that this is always appropriate. If an agent running a probabilistic coordinated attack algorithm that is guaranteed to work with high probability over the runs finds itself in a state where it knows that the attack will not be coordinated, then it seems clear that it should not proceed with the attack. It may be worth reconsidering a number of algorithms to see if they can be redesigned to give stronger guarantees. This may be particularly appropriate in the context of zero-knowledge protocols [GMR89], where the current definitions allow a prover to continue playing against a verifier even when the prover knows perfectly well that it has already leaked information to the verifier, and may continue to do so. Although it is extremely unlikely that the prover will find itself in this situation, it may be worth trying to redesign the protocol to deal with this possibility. While *adaptive protocols*,

[16] Although it is not clear from the definition of probabilistic attack given in [FZ88] over what the probability is being taken, the results given clearly assume that the probability is being taken over the runs.

where processors modify their actions in light of what they have learned, are common in the control theory literature, the probabilistic algorithms that are used in distributed systems typically are not adaptive. It seems that a number of algorithms can be converted to adaptive algorithms with relatively little overhead. We hope to study this issue more carefully in the future.

4.8 Conclusion

We have provided a framework for capturing knowledge and probability in distributed systems. Our framework makes it clear that in order for an agent to evaluate the probability of a formula φ at a given point, we need to specify the adversary (or, more accurately, adversaries) that determines the probability space. We have described how to choose the appropriate probability space as a function of the adversary, making no assumptions about the strategy the adversary is following. One potentially fruitful line of research is to understand how our results are effected if we make assumptions about the strategies the adversary p_j is allowed to follow (such as assuming that p_j is trying to maximize its payoff).

This use of adversaries may help clear up a number of subtle issues in the study of probability, such as what the probability that a coin lands heads is after the coin has been tossed. In addition, our approach allows us to unify the different approaches to probability in distributed systems that have appeared in earlier works. Of course, what needs to be done now is to use these definitions to analyze probabilistic (especially cryptographic) protocols!

4.A Proofs of results

This appendix contains the proofs of all results claimed in the chapter.

Proposition 4.1: If $S_{i,c}$ is state-generated and satisfies REQ_1, then $S_{i,c}$ satisfies REQ_2.

Proof: Given a global state g, let G_g be the set of points (r, k) with $r(k) = g$, and let $\mathcal{R}_g$ be the set of runs through g. By our technical assumption that the global state encodes the adversary, each global state is contained in precisely one computation tree. Thus, G_g and $\mathcal{R}_g$ are contained in a single

computation tree, and $\mathcal{R}_g = \mathcal{R}(G_g)$. Since $\mathcal{S}$ is state-generated, $S_{i,c}$ is the union of a collection of sets of the form G_g. Since $S_{i,c}$ satisfies REQ_1, it is contained in a single computation tree $\mathcal{T}_A = (\mathcal{R}_A, \mathcal{X}_A, \mu_A)$; and since a single computation tree contains at most a countable number of global states, $S_{i,c}$ is a countable union of sets of the form G_g. Thus, $\mathcal{R}(S_{i,c})$ is the countable union of sets of the form $\mathcal{R}_g = \mathcal{R}(G_g)$ with g a global state in $\mathcal{T}_A$. By the definition of $\mathcal{T}_A$, each set $\mathcal{R}_g$ is a measurable set of runs with positive measure, and hence their countable union $\mathcal{R}(S_{i,c})$ must also be a measurable set with positive measure. It follows that $S_{i,c}$ satisfies REQ_2. □

Proposition 4.2: If $S_{i,c}$ satisfies REQ_1 and REQ_2, then $P_{i,c}$ is a probability space.

Proof: We must show (see [Hal50]) that $\mathcal{X}_{i,c}$ is a set of subsets of $S_{i,c}$ including $S_{i,c}$ that is closed under the formation of complements and countable unions, and that $\mu_{i,c}$ is a nonnegative, countably additive function on $\mathcal{X}_{i,c}$ satisfying $\mu_{i,c}(\emptyset) = 0$.

Let $\mathcal{T}(c) = (\mathcal{R}_A, \mathcal{X}_A, \mu_A)$. Since $S_{i,c} = Proj(\mathcal{R}_A, S_{i,c})$ and $\mathcal{R}_A \in \mathcal{X}_A$, we have $S_{i,c} \in \mathcal{X}_{i,c}$. If $X \in \mathcal{X}_{i,c}$, then $X = Proj(R, S_{i,c})$ for some $R \in \mathcal{X}_A$; since $\mathcal{X}_A$ is closed under complementation, $R^c \in \mathcal{X}_A$ and $X^c = Proj(R^c, S_{i,c}) \in \mathcal{X}_{i,c}$, and hence $\mathcal{X}_{i,c}$ is closed under complementation. If $X_1, X_2, \ldots$ is a countable collection of sets from $\mathcal{X}_{i,c}$, then $X_j = Proj(R_j, S_{i,c})$ for some $R_j \in \mathcal{X}_A$ for each j. Since $\mathcal{X}_A$ is closed under countable union, $R = \cup_j R_j \in \mathcal{X}_A$. It follows that

$$X = \cup_j X_j = \cup_j Proj(R_j, S_{i,c}) = Proj(\cup_j R_j, S_{i,c}) = Proj(R, S_{i,c}),$$

so $X \in \mathcal{X}_{i,c}$ and $\mathcal{X}_{i,c}$ is closed under countable union.

Since $S_{i,c}$ is contained in a single computation tree by REQ_1, and since $\mathcal{R}(S_{i,c}) \in \mathcal{X}_A$ and $\mu_A(\mathcal{R}(S_{i,c})) > 0$ by REQ_2, conditional probability with respect to $\mathcal{R}(S_{i,c})$ is well-defined, and hence $\mu_{i,c}$ is well-defined. Clearly, $\mu_{i,c}$ is nonnegative since μ_A is. Furthermore, $\mu_{i,c}(\emptyset) = \mu_A(\emptyset)/\mu_A(\mathcal{R}(S_{i,c})) = 0$. Finally, suppose $X_1, X_2, \ldots$ is a countable collection of pairwise-disjoint sets in $\mathcal{X}_{i,c}$. We know that $X_j = Proj(R_j, S_{i,c})$ for some $R_j \in \mathcal{X}_A$. We can assume every run in R_j passes through $S_{i,c}$, or we can replace R_j with the measurable set $\mathcal{R}(S_{i,c}) \cap R_j$; and we can assume the R_j are pairwise disjoint, since if r is contained in both R_j and R_k then some point on r is contained in both

$X_j = Proj(R_j, S_{i,c})$ and $X_k = Proj(R_k, S_{i,c})$, contradicting the pairwise-disjointness of X_j and X_k. It follows from the pairwise-disjointness of the $R_j = \mathcal{R}(X_j)$ that

$$\begin{aligned}
\mu_{i,c}(\cup_j X_j) &= \frac{\mu_A(\mathcal{R}(\cup_j X_j))}{\mu_A(\mathcal{R}(S_{i,c}))} = \frac{\mu_A(\cup_j \mathcal{R}(X_j))}{\mu_A(\mathcal{R}(S_{i,c}))} \\
&= \sum_j \frac{\mu_A(\mathcal{R}(X_j))}{\mu_A(\mathcal{R}(S_{i,c}))} \\
&= \sum_j \mu_{i,c}(X_j),
\end{aligned}$$

and hence $\mu_{i,c}$ is countably additive. □

Proposition 4.3: In a synchronous system, if $\mathcal{S}$ is a consistent standard assignment and $\mathcal{L}(\Phi)$ is state-generated, then φ is measurable with respect to $\mathcal{S}$ for all facts $\varphi \in \mathcal{L}(\Phi)$.

Proof: Recall that $\mathcal{L}(\Phi)$ is state-generated if all the primitive propositions in Φ are facts about the global state. Recall also that φ is measurable with respect to $\mathcal{S}$ if $S_{i,c}(\varphi) \in \mathcal{X}_{i,c}$ for all agents p_i and points c. Fix an agent p_i and a point c. Let S_k denote the set of time k points in the computation tree containing c. We claim it is enough to show that

(*) $\mathcal{R}(S_k(\varphi))$ is a measurable set of runs for all times k and all formulas $\varphi \in \mathcal{L}(\Phi)$.

To see this, notice that since $\mathcal{S}$ is a consistent assignment in a synchronous system, $S_{i,c}$ contains only time k points for some k. Consequently, we have $\mathcal{R}(S_{i,c}(\varphi)) = \mathcal{R}(S_{i,c}) \cap \mathcal{R}(S_k(\varphi))$. Since $\mathcal{R}(S_{i,c})$ is measurable by REQ_2, condition (*) will imply $\mathcal{R}(S_{i,c}(\varphi))$ is measurable. It will follow that $S_{i,c}(\varphi)$ is a measurable subset of $S_{i,c}$.

The proof of (*) proceeds by induction on the structure of φ. If φ is a primitive proposition in Φ, then since $\mathcal{L}(\Phi)$ is state-generated we know that φ must be a fact about the global state. Arguments similar to those used for Proposition 4.1, therefore, suffice to show that $\mathcal{R}(S_k(\varphi))$ is a measurable set of runs. The cases of negation and conjunction follow immediately from the fact that measurable sets are closed under negation and intersection. Since

$K_i\varphi$ is a fact about the global state, the arguments for such a formula is identical to the argument for primitive propositions above.

For a probability formula ψ of the form $Pr_i(\varphi) \geq \alpha$, since we consider only uniform sample space assignments, it is easy to check that ψ is true at either all or none of the points in $S_{i,c}$; hence $\mathcal{R}(S_{i,c}(\psi))$ must be measurable since $\mathcal{R}(S_{i,c})$ itself is guaranteed to be measurable by REQ_2. Since $\mathcal{S}$ is inclusive, we know that $d \in S_{i,d}$ for every time k point d. Since $\mathcal{S}$ is consistent, we know that $S_{i,d}$ contains only time k points from $\mathcal{T}(d)$. It follows that S_k is the union of sets of time k points of the form $S_{i,d}$. Moreover, since $\mathcal{S}$ is uniform, the $S_{i,d}$ actually partition S_k. Finally, since each $S_{i,d}$ is state-generated and since there are at most a countable number of time k global states in any given tree, we see that S_k is partitioned into a countable collection of sets of the form $S_{i,d}$, and hence the same is true for $S_k(\psi)$. It follows that $\mathcal{R}(S_k(\psi))$ is partitioned into a countable collection of sets of the form $\mathcal{R}(S_{i,d})$, and since the sets $\mathcal{R}(S_{i,d})$ are measurable, so is their countable union $\mathcal{R}(S_k(\psi))$.

For $\bigcirc\varphi$, notice that φ is true at $(r, k+1)$ iff $\bigcirc\varphi$ is true at (r, k). It follows that $\mathcal{R}(S_k(\bigcirc\varphi)) = \mathcal{R}(S_{k+1}(\varphi))$, and hence by the inductive hypothesis for φ that $\mathcal{R}(S_k(\bigcirc\varphi))$ is a measurable set of runs. In fact, a simple extension of this argument (by induction on ℓ) shows that if $\mathcal{R}(S_k(\varphi))$ is measurable then so is $\mathcal{R}(S_k(\bigcirc^\ell\varphi))$.

For $\varphi\, U\, \psi$, define $\varphi\, U_0\, \psi$ to be the formula ψ, and define $\varphi\, U_\ell\, \psi$ for $\ell > 0$ to be the formula $\varphi \wedge \ldots \wedge \bigcirc^{\ell-1}\varphi \wedge \bigcirc^\ell\psi$. It is easy to see that $\varphi\, U\, \psi$ is true at a point d iff $\varphi\, U_\ell\, \psi$ is true at d for some $\ell \geq 0$. Thus, $S_k(\varphi\, U\, \psi) = \bigcup_{\ell\geq 0} S_k(\varphi\, U_\ell\, \psi)$ and hence $\mathcal{R}(S_k(\varphi\, U\, \psi)) = \bigcup_{\ell\geq 0} \mathcal{R}(S_k(\varphi\, U_\ell\, \psi))$. Since the induction hypothesis holds for the subformulas φ and ψ, the preceding paragraph shows that each set $\mathcal{R}(S_k(\varphi\, U_\ell\, \psi))$ is also measurable, and hence so is their countable union $\mathcal{R}(S_k(\varphi\, U\, \psi))$. □

Proposition 4.4: If $\mathcal{S}$ and $\mathcal{S}'$ are standard assignments satisfying $\mathcal{S}' \leq \mathcal{S}$, then for every agent p_i and point c, the set $S_{i,c}$ can be partitioned into sets of the form $S'_{i,d}$ with $d \in S_{i,c}$.

Proof: Suppose that $\mathcal{S}$ and $\mathcal{S}'$ are standard assignments satisfying $\mathcal{S}' \leq \mathcal{S}$. Since $\mathcal{S}'$ is inclusive, we have $d \in S'_{i,d} \subseteq S_{i,d} = S_{i,c}$ for every $d \in S_{i,c}$, and hence $S_{i,c}$ is the union of the $S'_{i,d}$ with $d \in S_{i,c}$. Furthermore, since $\mathcal{S}'$ is uniform, two sets $S'_{i,d}$ and $S'_{i,e}$ are either equal or disjoint, and hence $S_{i,c}$ can be partitioned into sets of the form $S'_{i,d}$ with $d \in S_{i,c}$. □

Proposition 4.5: In a synchronous system, if $\mathcal{P}'$ and $\mathcal{P}$ are consistent, standard assignments satisfying $\mathcal{P}' \leq \mathcal{P}$, then for all agents p_i, all points c, and all measurable subsets $S' \in \mathcal{X}'_{i,c}$

(a) $S' \in \mathcal{X}_{i,c}$ (so that, in particular, $S'_{i,c}$ itself is a measurable subset of $S_{i,c}$),

(b) $\mu_{i,c}(S'_{i,c}) > 0$,

(c) $\mu'_{i,c}(S') = \mu_{i,c}(S'|S'_{i,c}) = \frac{\mu_{i,c}(S')}{\mu_{i,c}(S'_{i,c})}$.

Proof: Fix an agent p_i, a time k point c of $\mathcal{T}_A = (\mathcal{R}_A, \mathcal{X}_A, \mu_A)$, and a set $S' \in \mathcal{X}'_{i,c}$.

(a) Since $S' \in \mathcal{X}'_{i,c}$, there must exist some subset $\mathcal{R}' \in \mathcal{X}_A$ such that $S' = Proj(\mathcal{R}', S'_{i,c})$. Without loss of generality, we can assume that $\mathcal{R}' \subseteq \mathcal{R}(S'_{i,c})$ (since we can replace $\mathcal{R}'$ with $\mathcal{R}' \cap \mathcal{R}(S'_{i,c})$, which must also be measurable since REQ_2 guarantees $\mathcal{R}(S'_{i,c})$ is measurable). Since $S'_{i,c} \subseteq S_{i,c}$ and both $S'_{i,c}$ and $S_{i,c}$ consist of time k points (since $\mathcal{P}$ and $\mathcal{P}'$ are consistent assignments), we have

$$\begin{aligned} Proj(\mathcal{R}', S'_{i,c}) &= \{(r', k) \in S'_{i,c} : r \in \mathcal{R}'\} = \{(r', k) \in S_{i,c} : r \in \mathcal{R}'\} \\ &= Proj(\mathcal{R}', S_{i,c}). \end{aligned}$$

Thus $S' = Proj(\mathcal{R}', S_{i,c})$, which shows that S' is a measurable subset of $S_{i,c}$.

(b) By part (a), it follows that $S'_{i,c}$ is a measurable subset of $S_{i,c}$. Since we have restricted to standard assignments S', we know that $S'_{i,c}$ is state-generated, and arguments similar to the proof of Proposition 4.1 show that $\mu_{i,c}(S'_{i,c}) > 0$.

(c) Tracing through definitions, we see

$$\begin{aligned} \mu'_{i,c}(S') &= \frac{\mu_A(\mathcal{R}(S'))}{\mu_A(\mathcal{R}(S'_{i,c}))} = \frac{\mu_A(\mathcal{R}(S'))/\mu_A(\mathcal{R}(S_{i,c}))}{\mu_A(\mathcal{R}(S'_{i,c}))/\mu_A(\mathcal{R}(S_{i,c}))} \\ &= \frac{\mu_{i,c}(S')}{\mu_{i,c}(S'_{i,c})} \\ &= \mu_{i,c}(S'|S'_{i,c}). \end{aligned}$$

□

Proposition 4.6: In a synchronous system, for all facts φ, all agents p_i, and all points c, the rule $Bet(\varphi, \alpha)$ is $Tree_{i,c}$-safe for p_i at c iff $Bet(\varphi, \alpha)$ is $Tree^j_{i,c}$-safe for p_i at c.

Proof: Since $\mathcal{P}^j \leq \mathcal{P}^{post}$, the sample space $Tree_{i,c}$ can be partitioned into the sample spaces $Tree^j_{i,d}$ with $d \in Tree_{i,c}$, and each such $Tree^j_{i,d}$ is a measurable subset of $Tree_{i,c}$ by Proposition 4.5. The law of conditional expectation, therefore, states that

$$E_{Tree_{i,c}}[W_f] = \sum E_{Tree_{i,c}}[W_f | Tree^j_{i,d}]\, \mu_{i,c}(Tree^j_{i,d}),$$

where the summation is taken over all sets of the form $Tree^j_{i,d}$ contained in $Tree_{i,c}$. Since $\mathcal{P}^j \leq \mathcal{P}^{post}$, we can use part (c) of Proposition 4.5 to prove that $E_{Tree_{i,c}}[W_f | Tree^j_{i,d}] = E_{Tree^j_{i,d}}[W_f]$, and hence that

$$E_{Tree_{i,c}}[W_f] = \sum E_{Tree^j_{i,d}}[W_f]\, \mu_{i,c}(Tree^j_{i,d}).$$

Suppose $Bet(\alpha, \varphi)$ is $Tree^j_{i,c}$-safe for p_i at c. Then $E_{Tree^j_{i,d}}[W_f] \geq 0$ for all points d agent p_i considers possible at c and all f, which implies $E_{Tree_{i,e}}[W_f] \geq 0$ for all points e agent p_i considers possible at c and all f, and hence that $Bet(\alpha, \varphi)$ is $Tree_{i,c}$-safe for p_i at c.

Conversely, suppose $Bet(\alpha, \varphi)$ is *not* $Tree^j_{i,c}$-safe for p_i at c. Then $E_{Tree^j_{i,d}}[W_f] < 0$ for some point d agent p_i considers possible at c and some f. Let f' be the strategy identical to f on $Tree_{j,d}$, and hence on $Tree^j_{i,d}$, but offering a payoff of 1 everywhere else. If p_j uses strategy f', there is clearly no way for p_i to win off of $Tree_{j,d}$ (the best p_i can do is break even), so that $E_{Tree^j_{i,e}}[W_{f'}] \leq 0$ for $e \neq d$. Moreover, by choice of d, $E_{Tree^j_{i,d}}[W_{f'}] < 0$. It follows that $E_{Tree_{i,c}}[W_{f'}] < 0$, and hence that $Bet(\alpha, \varphi)$ is *not* $Tree_{i,c}$-safe for p_i at c. □

Theorem 4.7: For all facts φ measurable with respect to $\mathcal{P}^j$, all agents p_i, and all points c, the rule $Bet(\varphi, \alpha)$ is safe for p_i at c iff $\mathcal{P}^j, c \models K_i^\alpha \varphi$.

Proof: Consider the evaluation of $E_c[W_f] = E_{Tree^j_{i,c}}[W_f(\varphi, \alpha)]$ for arbitrary points c and strategies f. Since p_j has the same local state at all points of $Tree^j_{i,c}$ and f is a function of p_j's local state, p_j offers the same payoff β for a bet on φ at all points of $Tree^j_{i,c}$. Since p_i is following $Bet(\varphi, \alpha)$ at all points

of $Tree^j_{i,c}$, agent p_i accepts the bet at all points of $Tree^j_{i,c}$ or rejects the bet at all such points, depending on whether $\beta \geq 1/\alpha$. If p_i rejects, then $E_c[W_f]$ is obviously 0. If p_i accepts, then p_i's profit is $\beta - 1$ at points satisfying φ and -1 at all other points, and hence $E_c[W_f] = \beta\mu^j_{i,c}(Tree^j_{i,c}(\varphi)) - 1$. (Notice that because φ is measurable with respect to $\mathcal{P}^j$, we are guaranteed that $Tree^j_{i,c}(\varphi)$ is a measurable subset of $Tree^j_{i,c}$, and hence $\mu^j_{i,c}(Tree^j_{i,c}(\varphi))$ is well-defined.)

Suppose $\mathcal{P}^j, c \models K^\alpha_i\varphi$. This means that $\mu^j_{i,d}(Tree^j_{i,d}(\varphi)) \geq \alpha$ for all points d agent p_i considers possible at c. For every point d agent p_i considers possible at c and every strategy f for p_j, therefore, we have $E_d[W_f] \geq 0$ since $\beta\mu^j_{i,d}(Tree^j_{i,d}(\varphi)) - 1 \geq (1/\alpha)\alpha - 1 = 0$ when $\beta \geq 1/\alpha$. It follows that $Bet(\varphi, \alpha)$ is safe for p_i at c.

Suppose $\mathcal{P}^j, c \not\models K^\alpha_i\varphi$. This means that $\mu^j_{i,d}(Tree^j_{i,d}(\varphi)) < \alpha$ for some point d agent p_i considers possible at c. Let f be the strategy for p_j offering a payoff of $1/\alpha$ for a bet on φ at all points p_j considers possible at d, and hence at all points of $Tree^j_{i,d}$, and 1 elsewhere. It follows that $E_d[W_f] < (1/\alpha)\alpha - 1 = 0$ for the given strategy f and the given point d agent p_i considers possible at c, and hence that $Bet(\varphi, \alpha)$ is not safe for p_i at c. □

Theorem 4.8: In a synchronous system, if $\mathcal{S}$ is a consistent standard assignment, then

(a) if $\mathcal{S} \leq \mathcal{S}^j$, then $\mathcal{S}$ determines safe bets against p_j, and

(b) if $\mathcal{S}$ determines safe bets against p_j and $\mathcal{L}(\Phi)$ is sufficiently rich, then $\mathcal{S} \leq \mathcal{S}^j$.

Proof: Theorem 4.7 tells us that $\mathcal{S}^j$ determines safe bets; from Theorem 4.9(a) (proved below), it follows that if $\mathcal{S} \leq \mathcal{S}^j$, then $\mathcal{S}$ determines safe bets too. This proves part (a).

To prove part (b), suppose $\mathcal{S} \not\leq \mathcal{S}^j$, which means $S_{i,c} \not\subseteq Tree^j_{i,c}$ for some agent p_i and point c. It is easy to construct a transition probability assignment τ inducing a distribution μ on the runs of $\mathcal{T}$ satisfying $\mu(\mathcal{R}(S_{i,c})) > \mu(\mathcal{R}(Tree^j_{i,c}))$. To see this, notice that $S_{i,c} \not\subseteq Tree^j_{i,c}$ implies $d \in S_{i,c}$ and $d \notin Tree^j_{i,c}$ for some time k point d in $\mathcal{T}$; and if G_d is the set of points with d's global state, then $G_d \subseteq S_{i,c}$ and $G_d \cap Tree^j_{i,c} = \emptyset$ since $\mathcal{S}$ and $\mathcal{S}^j$ are state-generated (they are standard). By causing τ to assign high probabilities to the edges in the path from the root of $\mathcal{T}$ to d's global state in $\mathcal{T}$, we can guarantee that $\mu(\mathcal{R}(G_d)) > 1/2$. This guarantees

that $\mu(\mathcal{R}(S_{i,c})) \geq \mu(\mathcal{R}(G_d)) > 1/2$, and since G_d and $Tree^j_{i,c}$ are disjoint, that $\mu(\mathcal{R}(Tree^j_{i,c})) < 1 - \mu(\mathcal{R}(G_d)) < 1/2$; so $\mu(\mathcal{R}(S_{i,c})) > \mu(\mathcal{R}(Tree^j_{i,c}))$ as desired.

Now let $\mathcal{P}$ be the probability assignment induced by $\mathcal{S}$ and τ, and let $\mathcal{P}'$ be the probability assignment induced by $\mathcal{S}'$ and τ. Furthermore, let G_c be the set of points with global state c, let ψ be the fact which is true precisely of the points in G_c, and let $\varphi = \neg\psi$. Since $\mathcal{L}(\Phi)$ is sufficiently rich, it follows that $\psi \in \Phi$; since $\mathcal{L}(\Phi)$ is closed under negation, it follows that $\varphi = \neg\psi \in \mathcal{L}(\Phi)$.

Since both $\mathcal{S}$ and $\mathcal{S}^j$ are standard, and hence both inclusive and state-generated, it follows that $G_c \subseteq S_{i,c} \cap Tree^j_{i,c}$. Since φ is false only at points in G_c, and since G_c is contained in both $S_{i,c}$ and $Tree^j_{i,c}$, it is easy to see that

$$\alpha = \mu_{i,c}(S_{i,c}(\varphi)) = \frac{\mu(\mathcal{R}(S_{i,c})) - \mu(\mathcal{R}(G_c))}{\mu(\mathcal{R}(S_{i,c}))}$$

and

$$\beta = \mu^j_{i,c}(Tree^j_{i,c}(\varphi)) = \frac{\mu(\mathcal{R}(Tree^j_{i,c})) - \mu(\mathcal{R}(G_c))}{\mu(\mathcal{R}(Tree^j_{i,c}))}.$$

Furthermore, since $\mathcal{S}$ is uniform (it is standard), any set $S_{i,e}$ not equal to $S_{i,c}$ is disjoint from $S_{i,c}$ and hence from G_c, so $\mu_{i,e}(S_{i,e}(\varphi)) = \mu_{i,e}(S_{i,e}) = 1$ for all such sets $S_{i,e}$. It follows that $\mathcal{P}, c \models K^\alpha_i \varphi$.

On the other hand, since $\mu(\mathcal{R}(S_{i,c})) > \mu(\mathcal{R}(Tree^j_{i,c}))$ and since $\mu(\mathcal{R}(G_c)) > 0$, it is easy to see that $\alpha > \beta$. Let f be the strategy in which p_j offers a payoff of $1/\alpha$ on $Tree_{j,c}$, and suppose p_i uses the rule $Bet(\varphi, \alpha)$. Clearly $W_f = W_f(\varphi, \alpha)$ is $1/\alpha - 1$ on $Tree^j_{i,c}(\varphi)$ and -1 off this set. Thus,

$$E(W_f) = \left(\frac{1}{\alpha} - 1\right)\beta + (-1)(1-\beta) < \left(\frac{1}{\alpha} - 1\right)\alpha - (1-\alpha) = 0,$$

which means $Bet(\varphi, \alpha)$ is not safe for p_i at c. □

Note that the universal quantification over transition probability assignments is crucial in this proof. Given a fact φ false only at points in the intersection of $S_{i,c}$ and $Tree^j_{i,c}$, the proof shows that a necessary condition for $\mathcal{P}, c \models K^\alpha_i \varphi$ to imply $Bet(\varphi, \alpha)$ is safe for p_i at c is that the measure of the runs through $S_{i,c}$ is less than or equal to the measure of the runs through $Tree^j_{i,c}$. In fact, this is a sufficient condition as well. For a given τ it may be possible to construct a set $S_{i,c} \not\subseteq Tree^j_{i,c}$ satisfying this condition; but the only way to satisfy this condition for all τ is to take $S_{i,c} \subseteq Tree^j_{i,c}$.

Theorem 4.9: In a synchronous system, if $\mathcal{P}'$ and $\mathcal{P}$ are consistent standard assignments satisfying $\mathcal{P}' < \mathcal{P}$, then

(a) for every fact φ, every agent p_i, every point c, and all α, β with $0 \leq \alpha \leq \beta \leq 1$, we have

$$\mathcal{P}', c \models K_i^{[\alpha,\beta]}\varphi \text{ implies } \mathcal{P}, c \models K_i^{[\alpha,\beta]}\varphi,$$

(b) there exist a fact φ, an agent p_i, a point c, and α, β with $0 \leq \alpha \leq \beta \leq 1$ such that

$$\mathcal{P}', c \not\models K_i^{[\alpha,1]}\varphi \text{ and yet } \mathcal{P}, c \models K_i^{[\alpha,1]}\varphi$$
$$\mathcal{P}', c \not\models K_j^{[0,\beta]}\neg\varphi \text{ and yet } \mathcal{P}, c \models K_j^{[0,\beta]}\neg\varphi.$$

If $\mathcal{L}(\Phi)$ is sufficiently rich, then $\varphi \in \mathcal{L}(\Phi)$.

Proof: First we prove part (a). Suppose $\mathcal{P}', c \models K_i^{[\alpha,\beta]}\varphi$. This means $\alpha \leq \mu'_{i,d_*}(S'_{i,d}(\varphi)) \leq \mu'^{*}_{i,d}(S'_{i,d}(\varphi)) \leq \beta$ for all points $d \in \mathcal{K}_i(c)$. Choose $d \in \mathcal{K}_i(c)$. Since $\mathcal{P}'$ and $\mathcal{P}$ are consistent (and uniform) and satisfy $\mathcal{P}' \leq \mathcal{P}$, the set $S_{i,d}$ is the disjoint union of a collection of probability spaces $S'_{i,d_1}, \ldots, S'_{i,d_\ell}$ with $d_j \in S_{i,d} \subseteq \mathcal{K}_i(c)$, each a measurable subset of $S_{i,d}$. It follows that $S_{i,d}(\varphi)$ is the disjoint union of $S'_{i,d_1}(\varphi), \ldots, S'_{i,d_\ell}(\varphi)$. An easy computation shows that $\sum_j \mu_{i,d_*}(S'_{i,d_j}(\varphi)) \leq \mu_{i,d_*}(S_{i,d}(\varphi))$. Since $\mathcal{P}' \leq \mathcal{P}$, Proposition 4.5 shows that μ'_{i,d_j} can be obtained from $\mu_{i,d}$ by conditioning on S'_{i,d_j}. It follows that

$$\begin{aligned}
\mu_{i,d_*}(S'_{i,d_j}(\varphi)) &= \sup\left\{\mu_{i,d}(T') \,:\, T' \subseteq S'_{i,d_j}(\varphi),\ T' \in \mathcal{X}'_{i,d_j}\right\} \\
&= \sup\left\{\mu'_{i,d_j}(T')\mu_{i,d}(S'_{i,d_j}) \,:\, T' \subseteq S'_{i,d_j}(\varphi),\ T' \in \mathcal{X}'_{i,d_j}\right\} \\
&= \sup\left\{\mu'_{i,d_j}(T') \,:\, T' \subseteq S'_{i,d_j}(\varphi),\ T' \in \mathcal{X}'_{i,d_j}\right\} \mu_{i,d}(S'_{i,d_j}) \\
&= \mu'_{i,d_j\,*}(S'_{i,d_j}(\varphi))\, \mu_{i,d}(S'_{i,d_j}).
\end{aligned}$$

Combining the preceding statements, we have

$$\begin{aligned}
\alpha &= \sum_j \alpha\, \mu_{i,d}(S'_{i,d_j}) \\
&\leq \sum_j \mu'_{i,d_j\,*}(S'_{i,d_j}(\varphi))\, \mu_{i,d}(S'_{i,d_j}) \\
&= \sum_j \mu_{i,d_*}(S'_{i,d_j}(\varphi)) \\
&\leq \mu_{i,d_*}(S_{i,d}(\varphi))
\end{aligned}$$

A similar argument shows $\mu_{i,d}{}^*(S_{i,d}(\varphi)) \leq \beta$. Since these arguments hold for all $d \in \mathcal{K}_i(c)$, it follows that $\mathcal{P}, c \models K_i^{[\alpha,\beta]}\varphi$.

We now prove part (b). Since $\mathcal{P}' < \mathcal{P}$, it follows that $S_{i,c}$ contains two distinct sets $S'_{i,c}$ and $S'_{i,d}$ for some agent p_i and points c and d. Let ψ be the fact true at precisely the points in the set G_c of points with c's global state, and let $\varphi = \neg\psi$. Notice that since $\mathcal{P}'$ is standard and hence state-generated, G_c is contained in $S'_{i,c}$ and disjoint from $S'_{i,d}$. If $\mathcal{L}(\Phi)$ is sufficiently rich, then $\psi \in \Phi$, and hence $\varphi = \neg\psi \in \mathcal{L}(\Phi)$.

Since $G_c \subseteq S'_{i,c} \subset S_{i,c}$, the fact φ holds with probability 1 with respect to all probability spaces determined by $\mathcal{P}'$ and $\mathcal{P}$ except $S'_{i,c}$ and $S_{i,c}$. Since $\mathcal{P}' \leq \mathcal{P}$, Proposition 4.5 tells us that $\mu'_{i,c}$ can be obtained from $\mu_{i,c}$ by conditioning on $S'_{i,c}$. It is easy to see, therefore, that φ holds with probability

$$\alpha' = \mu'_{i,c}(S'_{i,c}(\varphi)) = \frac{\mu_{i,c}(S'_{i,c}) - \mu_{i,c}(G_c)}{\mu_{i,c}(S'_{i,c})}$$

with respect to $S'_{i,c}$, and probability

$$\alpha = \mu_{i,c}(S_{i,c}(\varphi)) = \frac{\mu_{i,c}(S_{i,c}) - \mu_{i,c}(G_c)}{\mu_{i,c}(S_{i,c})}$$

with respect to $S_{i,c}$. Since $\mu_{i,c}(S'_{i,c}) < \mu_{i,c}(S_{i,c}) = 1$, however, it is easy to see that $\alpha' < \alpha$. It follows that $\mathcal{P}, c \models K_i^{[\alpha,1]}\varphi$ but $\mathcal{P}', c \not\models K_i^{[\alpha,1]}\varphi$.

On the other hand, $\neg\varphi$ holds with probability 0 with respect to all probability spaces determined by $\mathcal{P}'$ and $\mathcal{P}$ except $S'_{i,c}$ and $S_{i,c}$. The fact $\neg\varphi$ holds with probability $1-\alpha'$ with respect to $S'_{i,c}$ and probability $1-\alpha$ with respect to $S_{i,c}$. Since $\alpha' < \alpha$, we have $1-\alpha < 1-\alpha'$; setting $\beta = 1-\alpha$, it follows that $\mathcal{P}, c \models K_i^{[0,\beta]}\neg\varphi$ but $\mathcal{P}', c \not\models K_i^{[0,\beta]}\neg\varphi$. □

Proposition 4.10: $\mathcal{P}^{post}, c \models K_i^{[\alpha,\beta]}\varphi$ iff $\mathcal{P}^{pts}, c \models K_i^{[\alpha,\beta]}\varphi$, for every fact φ, agent p_i, and point c.

Proof: Consider the adversary $A \in pts$ mapping an agent p_i and a point d to the set $S^A_{i,d}$ of points defined as follows: for every run r passing through $Tree^j_{i,d}$, there is a point $(r,k) \in Tree_{i,d}$ satisfying $\neg\varphi$ in $S^A_{i,d}$ if such a point exists, and an arbitrary point $(r,k) \in Tree_{i,d}$ if all such points satisfy φ. It is easy to see that the same set of runs pass through $S^A_{i,d}$ and $Tree^j_{i,d}$, and that a

run r passes through $S^A_{i,d}(\varphi)$ iff φ is true at all points of r contained in $Tree_{i,d}$. It follows that $S^A_{i,d}(\varphi)$ and $Tree_{i,d}(\varphi)$ have the same inner measure. On the other hand, consider an arbitrary adversary $B \in pts$ mapping p_i and d to the set $S^B_{i,d}$ (contained in $Tree^j_{i,d}$). Suppose the run r passes through $S^A_{i,d}(\varphi)$. It follows from the definition of $S^A_{i,d}(\varphi)$ that φ must hold at every point $(r,k) \in Tree^j_{i,d}$. Since $S^B_{i,d}$ must contain precisely one such point, r must pass through $S^B_{i,d}(\varphi)$ as well. It follows that the inner measure of $S^B_{i,d}(\varphi)$ must be at least the inner measure of $S^A_{i,d}(\varphi)$; and hence that the infimum (taken over all adversaries $B \in pts$) of $(\mu^B_{i,d})_*(S^B_{i,d}(\varphi))$ is precisely $(\mu^{post}_{i,d})_*(Tree^j_{i,d}(\varphi))$. A similar construction shows that the supremum (taken over all adversaries $B \in pts$) of $(\mu^B_{i,d})^*(S^B_{i,d}(\varphi))$ is precisely $(\mu^{post}_{i,d})^*(Tree^j_{i,d}(\varphi))$. Since these statements are true for all points d agent p_i considers possible at c, we have $\mathcal{P}^{pts}, c \models K_i^{[\alpha,\beta]}\varphi$ iff $\mathcal{P}^{post}, c \models K_i^{[\alpha,\beta]}\varphi$. □

4.B Discussion

In this appendix, we discuss a few issues related to observations made in this chapter.

4.B.1 The need for protocols

Although from a computer scientist's point of view, it seems quite natural to assume, as we do, that all agents in a system follow some kind of a protocol, protocols are not quite so standard in the probability theory literature. Interestingly, Shafer observes [Sha85] that it is necessary for us to think in terms of protocols if we are to make sense of "conditioning on everything an agent knows" as is done by $\mathcal{P}^{post}$. His argument, which we reproduce here, is based on *Freund's puzzle of the two aces* (see [Fre65]; other references are given in [Sha85]).

Consider a deck with four cards, the ace and deuce of hearts and spaces. After a fair shuffle of the deck, two cards are dealt to p_1. Now what is the probability, according to p_2, that p_1 holds both aces? First, notice that if A, B, C, and D denote the events that p_1 holds two aces, at least one ace, the ace of spaces, and the ace of hearts, respectively, then

$$Pr(A) = Pr(A \cap B) = Pr(A \cap C) = \frac{1}{6},\ Pr(B) = \frac{5}{6},\ Pr(C) = Pr(D) = \frac{1}{2}.$$

Suppose p_1 first says it holds an ace. Conditioning on this information, p_2 computes the probability p_1 holds both aces to be

$$Pr(A|B) = \frac{1/6}{5/6} = \frac{1}{5}.$$

As a result of learning p_1 holds at least one ace, the probability according to p_2 that p_1 holds both aces increases.

Suppose p_1 then says it holds the ace of spades. Conditioning on this additional information, p_2 computes the probability p_1 holds both aces to be

$$Pr(A|C) = \frac{1/6}{1/2} = \frac{1}{3}.$$

As a result of learning p_1 holds not just an ace of spades but actually holds the ace of spades, the probability according to p_2 that p_1 holds both aces increases even more. Similarly, $Pr(A|D) = 1/3$.

But is this second computation reasonable? When p_2 learns B, then p_2 knows that p_1 has either the ace of spades or the ace of hearts. When p_2 learns C, then p_2 knows that p_1 definitely has the ace of spades. Is it reasonable for the probability p_2 places on event A, that p_1 holds two aces, to increase from 1/5 to 1/3 simply as a result of learning which of the two aces p_1 has? It seems just as reasonable to argue that the information about which ace p_1 actually has is useless, and p_2's probability of A shouldn't change upon hearing that C (or D) holds.

As Shafer points out, the right way for p_2 to update its probability of A depends on what protocol the agents are following. If the agents had agreed p_1 would first reveal whether it held an ace, and then whether it held the ace of spades, then the increase seems reasonable: if p_1 says it holds an ace, then p_2's learning p_1 does *not* hold the ace of spades causes p_2's probability that p_1 holds both aces goes down to 0; so learning that p_1 *does* hold the ace of spades should make p_2's probability go up. On the other hand, if the agents were following a protocol whereby p_1 first reveals whether it has an ace, and then, if it does, reveals the suit of one of the aces it holds, choosing between hearts and spades at random if it has both aces, then p_2's probability should not change as a result of hearing that p_1 holds the ace of spades.[17] We leave it to the reader to construct the computation trees corresponding to the two

[17]Although Shafer does not mention this point, the need to assume that p_1 chooses

protocols described above, and to check that using $\mathcal{P}^{post}$, we do indeed get the right probabilities in each case. Again, the key point here is that we need the protocol to be completely specified in order to appropriately compute the conditional probabilities.

4.B.2 Safe bets and nonmeasurable facts

Recall that the statement of Theorem 4.7 says that for *measurable* facts, $\mathcal{P}^j$ determines safe bets against p_j. The condition of measurability is required in order for the use of expectation in the definition of a safe bet to make sense. Remember that $Bet(\varphi, \alpha)$ is safe for p_i at c if $E_d(W_f) = E_{Tree^j_{i,d}}(W_f(\varphi, \alpha))$ is nonnegative for all points d agent p_i considers possible at c, and for all strategies f for p_j. We computed in the proof of Theorem 4.7 that $E_d(W_f) = \beta\mu_{i,d}(S_{i,d}(\varphi)) - 1$, where β is the payoff offered by p_j in $S_{i,d}$ ($S_{i,d}$ was actually $Tree^j_{i,d}$). In order for $\mu_{i,d}(S_{i,d}(\varphi))$ to be well defined, however, $S_{i,d}(\varphi)$ must be a measurable subset of $S_{i,d}$, which means φ must be measurable.

In fact, Theorem 4.7 holds for nonmeasurable facts as well, but we must first give a meaningful definition of expectation for nonmeasurable events. The intuition behind the inner and outer measures μ_* and μ^* of a measure space $(S, \mathcal{X}, \mu)$ is that $\mu_*(S')$ and $\mu^*(S')$ give upper and lower bounds on the probability of S'; if S' is actually a measurable set, of course, these bounds are equal to the actual probability. This is made precise by a classical result [Hal50] which says that if $(S, \mathcal{X}', \nu)$ extends $(S, \mathcal{X}, \mu)$ (in that $\mathcal{X}' \supseteq \mathcal{X}$ and μ and ν agree on $\mathcal{X}$), then for all sets $X \in \mathcal{X}'$, we have $\mu_*(X) \leq \nu(X) \leq \mu^*(X)$. Moreover, the bounds described by the inner and outer measure are actually attainable, in that for all subsets $X \subseteq S$, there is a probability space $(S, \mathcal{X}', \nu)$ extending $(S, \mathcal{X}, \mu)$ such that $X \in \mathcal{X}'$ and $\nu(X) = \mu_*(X)$; a similar result holds in the case of outer measure.

We want to extend these ideas to expected value. More precisely, we would like to define a notions of inner expected value and outer expected value for a "nonmeasurable" random variable X which give, respectively, lower and upper bounds on what should be the expected value of X if we

between hearts and spades at random if it holds both aces is crucial here. For example, suppose p_2 always tells p_1 it holds the ace of hearts when it holds both aces. In this case, p_2's probability p_1 holds both aces should decrease to 0 when p_1 says it holds the ace of spades.

were to extend the measure space as above to make X measurable. This requires some work in general, but in the special case where X takes on only two values, it can be done in a straightforward way. If the two values taken on by X are x and y, with $x > y$, then we define the inner and outer expectations of a random variable X by

$$E_*(X) = x\mu_*(X = x) + y\mu^*(X = y) \text{ and}$$
$$E^*(X) = x\mu^*(X = x) + y\mu_*(X = y).$$

It is not hard to show that these definitions agree with the expected value if the set $X = x$ is measurable, and that these values are attainable if we extend the probability space in the right way to make $X = x$ measurable.

Notice that the random variable W_f in which we are interested in fact takes on only two values (depending on whether φ is true or false). Thus, applying these definitions, we get:

$$\begin{aligned} E_*(W_f) &= (\beta - 1)\mu_*(S_{i,d}(\varphi)) + (-1)\mu^*(S_{i,d}(\neg\varphi)) \\ &= (\beta - 1)\mu_*(S_{i,d}(\varphi)) - (1 - \mu_*(S_{i,d}(\varphi))) \\ &= \beta\mu_*(S_{i,d}(\varphi)) - 1, \end{aligned}$$

which looks very similar to the formula computed for measurable facts. Following the last two paragraphs of the proof of Theorem 4.7 using this formula, it is easy to see the rest of the proof holds, and hence that Theorem 4.7 is true using inner expectation in place of expectation in the definition of a safe bet.

Chapter 5

A Knowledge-Based Analysis of Zero Knowledge

In this chapter we study the relationship between knowledge and cryptography. In particular, we define notions of knowledge for use in the context of cryptography, and analyze interactive and zero knowledge proof systems in terms of these notions of knowledge.

5.1 Introduction

Much of our intuition concerning cryptography depends heavily on the concept of knowledge. For example, various methods of encryption [RSA78, GM84] allow two agents to communicate via encrypted messages knowing that other polynomial-time agents will know little or nothing about the contents of their communication (subject to certain complexity-theoretic assumptions). Just as we argue informally about distributed computation in terms of the knowledge processors have about their environment, the same is true of cryptography. In fact, the whole point of cryptography is either to transfer knowledge to or to withhold knowledge from various agents in a system. While our intuition concerning cryptography depends heavily on knowledge, researchers have yet to make this intuition precise in terms of

This chapter is joint work with Joe Halpern and Yoram Moses. An earlier version of this work appeared in *Proceedings of the 20th ACM Symposium on Theory of Computing* [HMT88].

formal definitions of knowledge. The purpose of this chapter is to develop definitions of knowledge that we hope will be useful in the general construction, analysis, and understanding of cryptographic protocols.

When developing such definitions, it is helpful to keep in mind concrete examples of cryptographic protocols. One class of protocols, the class of *interactive* and *zero knowledge proof systems* [GMR89], has received a great deal of attention from the cryptographic community. Loosely speaking, an interactive proof is a conversation between an infinitely powerful *prover* and a polynomial-time *verifier* in which the prover tries to convince the verifier that a certain fact φ is true, typically a fact of the form $x \in L$. The proof consists of a sequence of rounds in which the verifier asks the prover a question, and the prover answers the question. Loosely speaking, such a proof is said to be zero knowledge if the prover does not leak any "knowledge" to the verifier: that is, anything the verifier knows (or knows how to compute) at the end of the proof the verifier already knows at the beginning of the proof (with the exception, of course, of the fact φ being proven).

The reason these protocols have received so much attention is that they seem to be fundamental building blocks in the construction of other cryptographic protocols. To see why this is true, consider two agents p and q both of whom want to use a certain resource in the system, and suppose they agree to flip a coin to determine which of them gets to use the resource first. Since neither wants the other to be able to influence the outcome of the coin in its own favor, how should p and q go about flipping this coin?

One such coin flipping scheme based on oblivious transfer is given by Rabin in [Rab81] (see also [Blu, FMR84]). This coin flipping scheme consists of four steps:

1. Agent p first selects two distinct, odd primes and sends their product n to agent q.

2. Agent q then selects an integer x at random from the group Z_n^* of integers between 1 and n relatively prime to n, and sends x^2 to p.

It is not hard to show, since n is the product of two distinct, odd primes, that x^2 will have four distinct square roots of the form x, $-x$, y, and $-y$. Agent p is able to compute these square roots since it knows the factorization of n.

3. Agent p randomly chooses a square root of x^2 and sends it to q.

Given one of x or $-x$ and one of y or $-y$, it is not hard to show that the greatest common divisor of $x+y$ and n or of $x-y$ and n is a nontrivial divisor of n. Agent q can easily compute the greatest common divisor of two numbers.

4. Agent q computes a nontrivial divisor of n and sends it to p.

Agent q wins the coin flip iff it sends a nontrivial divisor of n to p.

Suppose that p and q are honest and follow this protocol exactly (that is, they do not cheat in any way). In this case, it is not hard to convince oneself that agent q wins the coin toss with probability exactly 1/2: roughly speaking, since p chooses the square root to send to q at random, with probability 1/2 agent p sends either y or $-y$, in which case q can compute a divisor of n and win the coin toss; and with probability 1/2 agent p sends either x or $-x$, in which case q gains no new information to help it compute a divisor of n and presumably loses the coin toss. If p or q cheat in some way during the protocol, however, then it is possible for q to win the coin toss with some probability other than 1/2. The protocol depends, for example, on the fact that the integer n constructed by p is really the product of two distinct, odd primes as required. Since it seems possible p could construct an n not of this form that would skew the outcome of the coin flip in p's favor, q should demand to be convinced that n is of the correct form before continuing with the coin flip. On the other hand, p does not want q to know any more about the factorization of n after being convinced n is of the right form, since this could skew the outcome of the coin flip in q's favor. If q can compute one of the prime factors after being convinced n is of the correct form, for example, then q can always win the coin toss. What we need here is a way for p to convince q that n is of the right form without giving q any additional information about n, and this is precisely what zero knowledge proof systems are designed to do.[1]

Because interactive and zero knowledge proof systems serve as building blocks in the design of cryptographic protocols, and because the concept of knowledge is so fundamental to our understanding of these proof systems, we choose to begin our study of knowledge and cryptography with interactive and zero knowledge proof systems. In this work, we will concentrate

[1]As shown in [FMR84], zero knowledge proofs can also be used to avoid problems arising when q tries to cheat.

on developing definitions of knowledge that let us formalize our intuition concerning such proof systems. The notions of knowledge most appropriate in this context, however, are far more subtle than the standard notions of knowledge used so often in the analysis of distributed computation (and, in particular, the notions defined in Chapter 2). Since cryptographic protocols are typically probabilistic protocols that guarantee only that correctness conditions are satisfied with high probability, definitions of knowledge such as probabilistic knowledge discussed in Chapter 4 that incorporate knowledge and probability will almost certainly be useful. More perplexing, however, is the fact that the computational power of agents in cryptographic systems is typically assumed to be restricted to polynomial-time. Recall that, according to the standard information-theoretic definition of knowledge, an agent is said to know all facts that follow from its local state, regardless of the computational complexity of determining that these facts hold. In the context of cryptography, however, the computational intractability of a problem is used to keep secret certain pieces information. Cryptography is concerned with what an agent can compute that it knows in polynomial time, and cryptographic protocols typically make guarantees such as no *polynomial-time* agent knows any more after eavesdropping on a conversation between two other agents than it did beforehand. In this context, the standard definition of knowledge is clearly inappropriate.

Our fundamental contribution is the definition of *practical knowledge*, which incorporates knowledge and probability with restrictions on agents' computational powers. This definition is based on the definition of *resource-bounded knowledge* given in [Mos88], which defines knowledge in terms of polynomial-time tests an agent can use to determine whether it knows a fact. Using the definition of practical knowledge, we characterize interactive proof systems in terms of a formal statement about knowledge. This statement essentially says "at the end of a proof of $x \in L$, the verifier knows $x \in L$," which is precisely what our intuition demands of an interactive proof system. Furthermore, using the definition of practical knowledge, we state a property of zero knowledge we call *knowledge security*, and prove that any zero knowledge proof system satisfies this property. Loosely speaking, this property says "the prover in a zero knowledge proof of $x \in L$ knows, with high probability, that if the verifier knows a fact φ at the end of the proof, then the verifier already knows $x \in L \supset \varphi$ at the beginning of the proof." This captures our intuition that a zero knowledge proof does not "leak"

knowledge of any fact other than facts following from $x \in L$, the fact the prover initially set out to prove.

Related to the concept of knowing a fact is knowing how to do something (how to perform a given operation). There is a difference, for example, between knowing *the fact* that an integer is composite and knowing *how to generate* a prime factor of the integer. Zero knowledge proofs are intended not to leak any knowledge of this kind as well as any knowledge of facts. While this concept of "knowing how" has also been of great interest in philosophy and AI (see [Moo85]), standard notions of knowledge do not capture this aspect of knowledge. We define a notion of knowing how to generate a y satisfying a relation $R(x, y)$, again incorporating knowledge and probability with bounds on agents' computational resources. In the context of a proof of $x \in L$, for example, we might take the relation $R(x, y)$ to mean "y is a prime factor of x." With this definition, we can again state a property of zero knowledge proof systems we call *generation security*, and prove that any zero knowledge proof system satisfies this property. This property essentially says "the prover in a zero knowledge proof of $x \in L$ knows, with high probability, that if the verifier knows how to generate a y satisfying $R(x, y)$ at the end of the proof, then the verifier knows how to do so at the beginning of the proof." This captures our intuition that during a zero knowledge proof the prover does not "leak" to the verifier any knowledge of how to do anything, let alone any knowledge of facts.

We find it interesting that, while these two properties (knowledge and generation security) capture everything the popular intuition says we want from zero knowledge proof systems, we are unable to prove that any proof system satisfying these properties is zero knowledge. This raises the interesting question of whether the cryptographic definition of a zero knowledge proof system is one of several possible implementations of what we should be calling zero knowledge, or whether there is some crucial aspect of this clever definition of zero knowledge the popular intuition is missing.

Other questions about zero knowledge proof systems also arise in this framework. For example, recall that interactive and zero knowledge proof systems are defined in the context of infinitely powerful provers, but only polynomial-time verifiers. In practice, however, *both* the prover and the verifier are polynomial-time agents. Although most of the proof systems defined in the context of infinitely powerful provers can be followed by polynomial-time provers if these weak provers are supplied with some secret information

(such as the factorization of n in the coin flipping example above), an interesting question to ask is whether any properties of these proof systems change as a result of the fact that the prover is a polynomial-time agent and not infinitely powerful. For example, suppose we are given an interactive proof system for membership in a language L defined in the context of infinitely powerful provers, and suppose we run this protocol in the context of weak provers. Is this protocol still a proof system for membership in L, or does it actually prove more or less than simple membership in L?

In order to answer such questions, we define *weak interactive proof systems* in which the prover (as well as the verifier) is restricted to probabilistic, polynomial-time computation. We prove that if L has a weak interactive proof system, then L must be contained in BPP (and hence that the verifier can determine whether $x \in L$ on its own without even consulting the prover). Since the interesting languages having proof systems in the context of infinitely powerful provers are not known to be contained in BPP (see [GMR89, GMW86]), these proof systems must prove more to the verifier than simple language membership when run by polynomial-time provers. In fact, we can prove in a precise sense that such proof systems must actually be proofs about the prover's knowledge. Furthermore, we show that, under natural conditions, the notions of interactive proofs of knowledge defined in [FFS87] and [TW87] are instances of such weak interactive proofs of knowledge. In this framework, using the language of knowledge, we can make precise several differences between these two notions of proofs of knowledge. Finally, we show that zero knowledge weak interactive proofs guarantee the same type of security with respect to the facts they prove as zero knowledge interactive proofs guarantee with respect to language membership.

We believe that our analysis provides a great deal of insight into (and support for) the definitions in [GMR89] and their extensions to the case of proofs about knowledge in [FFS87, TW87]. None of our technical results about the definitions themselves is very deep; the difficulty was in coming up with the right notions of knowledge to use when thinking about them. While the definitions of knowledge we give here are motivated by interactive and zero knowledge proof systems, we believe they are potentially useful when thinking about cryptographic protocols in general. We note that Fischer and Zuck [FZ87] also consider notions of knowledge (closely related to our notion of knowing how to generate) for use in the context of interactive and zero knowledge proof systems, and use their definitions of knowledge to analyze

an interactive proof of quadratic residuosity. We believe that thinking about interactive and zero knowledge proof systems (and cryptography in general) in terms of knowledge provides a good framework within which to think about cryptographic definitions and their appropriateness.

The rest of the chapter is organized as follows. In the next section, Section 5.2, we give the cryptographic definitions of interactive and zero knowledge proof systems. In Section 5.3, we show how these definitions motivate the the definition of *practical knowledge*. In the following Sections 5.4 and 5.5, we show how practical knowledge can be used to characterize interactive proof systems in terms of knowledge, and how practical knowledge can be used to make precise the intuition that the verifier in a zero knowledge proof does not know any more at the end of the proof than it did at the beginning. In Section 5.6 we define the notion of "knowing how," and show that, in a precise sense, the verifier cannot *do* any more at the end of a zero knowledge proof than it could at the beginning. Section 5.7 introduces weak interactive proofs, relates them to the proofs of knowledge of [FFS87, TW87], and proves that zero knowledge weak interactive proofs are secure in the senses defined above. Finally, in Section 5.8, having characterized the definition of an interactive proof system in terms of knowledge, we sketch an example of how we can use this characterization to reason about interactive proof systems. More precisely, we prove the familiar result that the sequential composition of two interactive proofs is itself an interactive proof. The chapter ends with Appendix 5.A, in which we give the proofs of the results claimed in this chapter.

5.2 Interactive and Zero Knowledge Proof Systems

We begin with the formal cryptographic definitions of interactive and zero knowledge proof systems, and a few informal examples.

5.2.1 Interactive protocols

Recall that, loosely speaking, an interactive proof is a conversation between a *prover* and a *verifier* in which the prover tries to convince the verifier that

a certain fact is true. This idea of a conversation between two agents is made precise by the definition of an interactive protocol.

Formally, an *interactive protocol* [GMR89] is an ordered pair (P, V) of probabilistic Turing machines, where P and V are intuitively descriptions of the protocols to be followed by the prover p and the verifier v, respectively. The Turing machines P and V share a read-only *input tape*; each has a private one-way, read-only *random tape*; each has a private *work tape*; and P and V share a pair of one-way *communication tapes*, one from P to V being write-only for P and read-only for V, and the other from V to P being write-only for V and read-only for P.

A *run* of the protocol (P, V) proceeds as follows. Initially, the common input tape is initialized with some string x, the two random tapes are initialized with infinite strings of independent, random bits, the two work tapes are initialized with strings s and t,[2] and the two communication tapes are blank.[3] The remainder of the run consists of a sequence of rounds. During any given round, V first performs some internal computation making use of its work tape and other readable tapes, and then sends a message to P by writing on V's write-only communication tape (which is P's read-only tape); P then performs a similar computation. It is not hard to see, for example, that we can view the coin flipping example given in the introduction as a two-round interactive protocol.

At any time during a run of an interactive protocol (P, V), either P or V can halt the interaction by entering a halt state. V can *accept* or *reject* an interaction by entering an accepting or rejecting halt state, respectively, in which case we refer to the resulting run as either an *accepting* or *rejecting run*. The running time of P or V during a run of (P, V) is the total number of steps taken by P or V, respectively, during the run. We assume that V is a probabilistic Turing machine running in time polynomial in $|x|$, and hence that it can perform only probabilistic, polynomial-time computations during each round, and participate in only a polynomial number of rounds. Consequently, we can assume that V always halts the interaction after a polynomial

[2] The the need for allowing initial values on the work tapes was first observed in [Ore87, TW87]; we will return to this issue when we define zero knowledge in Section 5.2.3 and weak interactive proof systems in Section 5.7.

[3] Actually, since we want to run interactive protocols as subroutines of other protocols, it is enough to assume the unread cells on the communication tapes, the cells to the right to the tape heads, are blank.

number of rounds, and always enters either an accepting or rejecting state. We will make no assumption about the running time of P for the moment, although in Section 5.7 (when we consider weak provers) we will assume that P runs in probabilistic, polynomial-time as well.

In terms of the model of computation defined in Chapter 2, the system corresponding to the interactive protocol (P, V) consists of two agents, the prover p and the verifier v. Notice that we distinguish the agents p and v from the protocols P and V they follow. A run of this interactive protocol is an infinite sequence of global states, where each global state consists of one local state for the prover p and one for the verifier v. Agent p's local state is a tuple consisting of a description of the Turing machine P, the current round number (an interactive protocol is a synchronous protocol), the contents of the input tape, the finite prefix of its random tape read up to this point, the contents of its work tape, the contents of the two communication tapes, and the position of the tape heads on each of these tapes; agent v's local state is defined in a similar fashion. We assume for the sake of convenience that prover and verifier each encode their complete history on their work tapes. Since we think of the prover and verifier as alternating steps, we think of the verifier as being active at even times, and the prover being active at odd times. It is not hard to see that the protocols described by the Turing machines P and V can be captured in terms of the definition of a protocol given in Chapter 2. We denote the system consisting of all possible runs of (P, V) by $P \times V$. The following systems will also be useful later in this chapter: $P \times \mathcal{V}^{pp}$, the system consisting of the union of the systems $P \times V^*$ for all probabilistic, polynomial-time V^*; $\mathcal{P} \times V$, the system consisting of the union of the systems $P^* \times V$ for all Turing machines P^*; and $\mathcal{P}^{pp} \times V$, the system consisting of the union of the systems $P^* \times V$ for all probabilistic, polynomial-time P^*.

5.2.2 Interactive proof systems

The next step in the definition of a zero knowledge proof system is to define what it means for an interactive protocol to be a proof system. Loosely speaking, an interactive protocol is a proof system for a language L if the verifier accepts the common input x with high probability when $x \in L$ and rejects with high probability when $x \notin L$.

Given an interactive protocol (P, V), we denote by $(P(s), V(t))(x)$ the

random variable assuming as values runs of the protocol (P, V) in which the input tape is initialized with x and the prover and verifier work tapes are initialized with s and t. More precisely, we denote by $(P(s), V(t))(x)$ the random variable mapping a sequence ρ of coin flips to the run of (P, V) in which the common input tape is initialized with x, the prover's work tape with s, and the verifier's work tape with t, and ρ is the sequence of coins flipped by the prover and verifier during this run.[4] We write '$(P(s), V(t))(x)$ *accepts*' to denote the fact that the run assumed by $(P(s), V(t))(x)$ as a value is an accepting run. An interactive protocol (P, V) is said to be an *interactive proof system for a language* L if the following conditions are satisfied:

- *Completeness*: For every $k \geq 1$ and sufficiently large x, and for every s and t,

$$\text{if } x \in L, \text{ then } pr[(P(s), V(t))(x) \text{ accepts}] \geq 1 - |x|^{-k}.$$

- *Soundness*: For every $k \geq 1$ and sufficiently large x, for every P^*, and for every s and t,

$$\text{if } x \notin L, \text{ then } pr[(P^*(s), V(t))(x) \text{ accepts}] \leq |x|^{-k}.$$

We use "sufficiently large x" as a shorthand for "there exists $N_k \geq 1$ such that for every x satisfying $|x| \geq N_k$." The subscript k in N_k reflects the fact that the notion of "sufficiently large" depends on the size of k. Without loss of generality, we can always assume that the same value N_k is used in both the soundness and completeness conditions.

We refer to p as the "good prover" when it is running P, and to v as the "good verifier" when it is running V. The completeness condition is a guarantee to both the good prover and the good verifier that if $x \in L$, then with overwhelming probability the good prover will be able to convince the good verifier that $x \in L$. The soundness condition is a guarantee to the good verifier that if $x \notin L$, then the probability that an arbitrary (possibly malicious) prover P^* is able to convince the good verifier that $x \in L$ is very

[4] We sometimes refer to a run assumed as a value by $(P(s), V(t))(x)$ as "a run of (P, V) on input x with s and t." We often abuse notation and use $(P(s), V(t))(x)$ to denote an arbitrary such run, or even the set of all such runs. The meaning will always be clear from context.

low. Intuitively, therefore, the verifier "knows" that $x \in L$ when it accepts, since the chance of accepting when $x \notin L$ is so low.

We note that this definition of an interactive proof system is stated in terms of a distribution over coin flips. This definition can be translated immediately into a statement in terms of a distribution over runs using the framework given in Chapter 4 as follows. Notice that once we fix the initial state (meaning that we fix P, V, s, t, and x), we can view the runs with this initial state as a single computation tree as defined in Chapter 4. Recall that $P \times V$ is the system consisting of all possible runs of (P, V), and that $\mathcal{P} \times V$ is the system consisting of the union of the systems $P^* \times V$ for all Turing machines P^*. In terms of the assignment $\mathcal{P}^{fst}$, the soundness condition says that the formula $Pr[\Diamond accept] \geq 1 - |x|^{-k}$ is true at all initial points of $P \times V$ satisfying $x \in L$, and the completeness condition says that the formula $Pr[\Diamond accept] \leq |x|^{-k}$ is true at all initial points of $\mathcal{P} \times V$ satisfying $x \notin L$. In this chapter, we are careful to write $pr[\varphi] \geq \alpha$ when the probability space is a set of coin flips, and to write $Pr[\varphi] \geq \alpha$ when the probability space is a set of runs (and, in particular, when $Pr[\varphi] \geq \alpha$ is to be interpreted as a formula in our logic of knowledge and probability).

One of the best known examples of an interactive proof system is the proof system for graph isomorphism from [GMW86]. Two graphs G_0 and G_1 are said to be *isomorphic* if there is a bijection h between the nodes of G_0 and G_1 with the property that (u, v) is an edge of G_0 iff $(h(u), h(v))$ is an edge of G_1. The graph isomorphism problem is formulated in terms of membership in the language of ordered pairs (G_0, G_1), where G_0 and G_1 are isomorphic graphs. One simple interactive proof system for graph isomorphism is for the prover, on input (G_0, G_1), to send the verifier an isomorphism h between G_0 and G_1, and have the verifier check that h is indeed an isomorphism; but this clearly gives the verifier more information than the simple fact that the two graphs are isomorphic: it actually gives the verifier an isomorphism! The protocol of [GMW86] is not this explicit. Suppose h is an isomorphism from G_0 to G_1, which can either be computed by an infinitely powerful prover or supplied as auxiliary input to the prover as an initial value on its work tape. The protocol consists of $n = |(G_0, G_1)|$ rounds, where each round consists of the following sequence of steps:

1. The prover

   (a) chooses a random permutation π of the vertices of $G_0 = (V_0, E_0)$,

(b) computes $H = (V_0, E)$, where H is the graph isomorphic to G_0 defined by $(\pi(u), \pi(v)) \in E$ iff $(u, v) \in E_0$, and

(c) sends H to the verifier.

2. The verifier chooses a bit α at random and sends α to the prover.

3. The prover sends the verifier an isomorphism from G_α to H: if $\alpha = 0$, the prover sends h; if $\alpha = 1$, the prover sends πh^{-1}.

4. The verifier checks that the mapping received from the prover is indeed an isomorphism from G_α to H.

The verifier accepts at the end of n rounds iff all n iterations of the protocol are successfully completed.

It is not hard to show that this interactive protocol is indeed an interactive proof system for graph isomorphism. If the two graphs G_0 and G_1 are isomorphic, then the prover will always be able to send the verifier an isomorphism π or πh^{-1} from G_0 or G_1 to H, depending on which is requested by the verifier, and hence will always cause the verifier to accept. Thus, the completeness condition is satisfied. If the two graphs G_0 and G_1 are *not* isomorphic, then the graph H sent to the verifier by the prover (by *any* prover, in fact) cannot be isomorphic to *both* G_0 and G_1, and the fact that the verifier chooses the bit α at random means that with probability 1/2 the verifier will ask the prover for an isomorphism between H and the graph to which H is *not* isomorphic, which the prover will certainly be unable to do. The probability, therefore, that a prover (any prover) will be able to supply the requested isomorphism on each iteration, and hence cause the verifier to accept incorrectly, is at most $1/2^n$. Thus, the soundness condition is satisfied, and the protocol is an interactive proof system for graph isomorphism.

This discussion shows that the verifier can use the protocol above to determine (with the prover's help) whether two graphs are isomorphic. It is not initially clear, however, that the verifier *cannot* use this protocol in some "unauthorized" way to determine whether some *other* fact is true. For example, suppose that the verifier chooses the α's in some way depending on the graphs H sent by the prover, rather than choosing the α's at random as required by the protocol. Is it possible for the verifier to use the protocol in this way, and then compute whether a certain value x is a quadratic residue modulo n, where n is the number of vertices in the two graphs; or then

determine whether one of G_0 or G_1 is isomorphic to a third graph G? The intuition behind zero knowledge is that such use of the protocol should be impossible.

5.2.3 Zero knowledge proof systems

This intuition that the verifier cannot use the graph isomorphism protocol to determine the truth of facts other than whether the two input graphs G_0 and G_1 are isomorphic is captured as follows. Loosely speaking, we say that an interactive proof system (P, V) is zero knowledge if, whenever $x \in L$, the verifier is able to generate on its own the conversations it could have had with the prover during an interactive proof of $x \in L$. Consequently, the fact $x \in L$ is the only knowledge gained by the verifier as a result of the proof of $x \in L$: if the verifier is able to determine the truth of some other fact after conversations with the prover, then the verifier is able to determine the truth of the fact on its own by generating these conversations on its own. In particular, if it is possible for the verifier to use the graph isomorphism protocol to determine whether one of G_0 or G_1 is isomorphic to a third graph G, then it is possible for the verifier to determine the truth of this fact on its own without even talking to the prover.

The intuition that the verifier can generate these conversations on its own is captured as follows. Consider runs of the protocol (P, V) with input x and work tapes s and t. From the verifier's point of view, a conversation with the prover (that is, a run) is uniquely determined by the verifier's *local history* of the run, where the verifier's local history is the sequence of local states the verifier assumes during the run. Intuitively, when we say that the verifier can generate on its own the conversations it has with the prover, we mean there is a Turing machine M that on input x and t generates local histories with the same distribution the verifier would see these local histories during runs of $(P(s), V(t))(x)$.[5]

[5]This intuition is formulated slightly differently in [GMR89]. They note that, given x and t, the verifier's view of a conversation with the prover is uniquely determined by the finite sequence ρ of random bits it uses during the conversation together with the finite sequence $\alpha_1, \ldots, \alpha_n$ of messages it receives from the prover; everything else the verifier sees during the conversation (e.g., the messages it sends) can be efficiently computed given this information. They call the tuple $(\rho, \alpha_1, \ldots, \alpha_n)$ the verifier's *view* of the run, and say that the verifier can generate on its own the conversations it has with the prover if there

This is made precise as follows (cf. [GMR89, GMW86, Ore87]). Suppose we have some domain *Dom* whose elements are of the form $(x, \bar{y})$, where x is a string and $\bar{y}$ is a vector of strings. Suppose for each $(x, \bar{y}) \in Dom$ we have two random variables $U_{x,\bar{y}}$ and $V_{x,\bar{y}}$ together with their associated probability distributions. The families $\{U_{x,\bar{y}} : (x, \bar{y}) \in Dom\}$ and $\{V_{x,\bar{y}} : (x, \bar{y}) \in Dom\}$ are said to be *perfectly indistinguishable* if the distributions of $U_{x,\bar{y}}$ and $V_{x,\bar{y}}$ are identical for all $(x, \bar{y}) \in Dom$.

Given an interactive protocol (P, V^*), we denote by $(P(s), \underline{V}^*(t))(x)$ the random variable assuming as values the verifier's local histories of runs of $(P(s), V^*(t))(x)$, where the distribution is determined by the coins flipped by the prover and the verifier. More precisely, we define $(P(s), \underline{V}^*(t))(x)$ to be the function mapping a sequence ρ of coin flips to the the verifier's local history of that run of $(P(s), V^*(t))(x)$ in which ρ is the sequence of coins flipped by the prover and the verifier. Given a probabilistic Turing machine M, we denote by $M(t, x)$ the random variable assuming as values the outputs generated by M on inputs t and x, where the distribution is determined by the coins flipped by M. An interactive proof system (P, V) for L is said to be *perfect zero knowledge* (cf. [GMR89]) if for every verifier V^* there is a probabilistic Turing machine M_{V^*} such that

1. $M_{V^*}(t, x)$ runs in *expected* time polynomial in $|x|$, and

2. the families

$$\{(P(s), \underline{V}^*(t))(x) : (x, s, t) \in Dom\} \text{ and } \{M_{V^*}(t, x) : (x, s, t) \in Dom\}$$

   are perfectly indistinguishable, where $(x, s, t) \in Dom$ iff $x \in L$, s is a possible input f·- P, and t is a possible input for V^*.

This definition says an interactive proof system is perfect zero knowledge if the verifier V^* can generate local histories on its own, using M_{V^*}, with precisely the same distribution it would see these local histories during runs of (P, V^*) on input x with s and t.

---

is a Turing machine M that on input x and t generates views with the same distribution the verifier would see these views during runs of (P, V) on input x with s and t. The two formulations are equivalent, of course, since the local history is efficiently computable from the view, and vice versa.

It is not too hard to show, for example, that the interactive protocol for graph isomorphism given above is actually zero knowledge [GMW86]. To see this, fix a verifier protocol V^*, and let us construct a simulating Turing machine M_{V^*} that generates local histories (one local state at a time) with the same distribution the verifier would observe during runs of (P, V^*). The Turing machine M_{V^*} is defined as follows: for each round $i = 1, \ldots, n$,

1. M_{V^*} first tries to guess the bit α_i that V^* will choose: M_{V^*} chooses a random bit β.

2. M_{V^*} then chooses a random permutation π_i of the nodes of G_β, and writes on V^*'s input communication tape the isomorphic copy H_i of G_β defined by $(\pi_i(u), \pi_i(v))$ is an edge of H_i iff (u, v) is an edge of G_β.

3. M_{V^*} simulates the Turing machine V^* until V^* writes a bit α_i on its output communication tape.

4. M_{V^*} reads α_i.

   (a) If $\alpha_i = \beta$, then M_{V^*} writes π_i on V^*'s input communication tape and outputs the verifier's local state. More precisely, M_{V^*} outputs three local local states: the state after the prover sends H_i, the state after the verifier sends α_i, and the state after the prover sends π_i.

   (b) If $\alpha_i \neq \beta$, then M_{V^*} rewinds V^* to its configuration at the beginning of this iteration (this includes erasing H_i from V^*'s input communication tape) and repeats steps 1-4.

The first key observation here is that, when the two graphs G_0 and G_1 are isomorphic, a random permutation of G_0 is a random permutation of G_1. It follows that the probability of generating H by choosing G_β at random and choosing a permutation of G_β at random is equal to the probability of generating H by choosing a permutation of G_0 at random. The second key observation is that, although M_{V^*} may have to try a number of times before it can finish the ith iteration and generate the graph H_i, the tries are independent. It follows that the conditional probability a graph H is generated on the kth try for the ith iteration, given that the first $k-1$ tries have failed, is the same for all k; and hence that the probability H is

generated by M_{V^*} on the ith iteration is equal to the probability the prover outputs H in the ith round. Since the remainder of the ith round is simply a simulation of (P, V^*), it follows that the distributions generated by (P, V^*) and M_{V^*} are identical. (We leave it to the reader to verify that the expected number of tries required for M_{V^*} to complete the ith iteration is 2, and hence that M_{V^*} runs in expected polynomial time; see [GMW86].)

The requirement that M_{V^*} generates local histories with precisely the same distribution with which they occur during runs of (P, V^*), however, is a very strong requirement. Since we are interested in what a polynomial-time verifier can learn as a result of a conversation with the prover, it should be sufficient if (cf. [GM84]) no polynomial-time test (meaning no test that can be used by a polynomial-time verifier) can detect any difference between the distributions generated by M_{V^*} and (P, V^*).

This intuition is formalized as follows (cf. [GMR89, GMW86, TW87, Ore87]). Two families $\{U_{x,\bar{y}} : (x, \bar{y}) \in Dom\}$ and $\{V_{x,\bar{y}} : (x, \bar{y}) \in Dom\}$ of random variables are said to be ***polynomially indistinguishable*** if for every probabilistic, polynomial-time algorithm M and every constant $k \geq 1$ there exists a constant $N_{M,k} \geq 1$ such that for all x with $|x| \geq N_{M,k}$ and all $\bar{y}$ with $(x, \bar{y}) \in Dom$ we have

$$|pr[M \text{ accepts } U_{x,\bar{y}}] - pr[M \text{ accepts } V_{x,\bar{y}}]| \leq |x|^{-k}.$$

It is important to notice that the probability is being taken over both the coin flips of M *and* the distributions of $U_{x,\bar{y}}$ and $V_{x,\bar{y}}$. It is also important to notice that the quantification over x (e.g., the common input) is not the same as the quantification over $\bar{y}$ (e.g., the auxiliary inputs to the prover and verifier).

The definition of what it means for an interactive proof system (P, V) for L to be (***polynomially***) ***zero knowledge*** is obtained by replacing perfect indistinguishability with polynomial indistinguishability in the definition of perfect zero knowledge. This definition of zero knowledge is actually the definition given in [GMW86] (and also in [Ore87]). This is the definition of zero knowledge we use in the remainder of this chapter. Other notions of zero knowledge based on other notions of indistinguishability (***statistical indistinguishability*** and ***computational indistinguishability***) are defined in [GMR89]. Since these notions of indistinguishability imply polynomial indistinguishability, and since our results are proven in the context of polynomial indistin-

guishability, our results (Theorems 5.5, 5.6, and 5.7) hold in the context of these other notions of indistinguishability as well.

5.3 Knowledge

With these examples in mind, we now define notions of knowledge for use in the analysis of cryptographic protocols. Among other things, these examples have two distinguishing features.

First, they are probabilistic. Correctness conditions (such as the soundness and completeness conditions for an interactive proof) guarantee that given properties hold with high probability, but not with certainty. Thus, while agents are justified in having a high degree of confidence that these properties hold, agents do not *know* they hold. Clearly, some definition of knowledge incorporating probability such as probabilistic knowledge defined in Chapter 4 will be useful here.

Second, and most important, the security of a zero knowledge protocol depends on the fact that the verifier's computational power is restricted to polynomial time, since the protocol's security depends on the fact that a polynomial-time agent cannot distinguish distributions on local histories generated by M_{V^*} and (P, V^*). In general, a common feature of cryptographic protocols is the use of computational intractability to keep information secret. While we are willing to accept the fact that an infinitely powerful verifier might be able to make unexpected use of a zero knowledge proof, we are not willing to accept the possibility a polynomial-time agent could increase its knowledge in the same way. To study such protocols in terms of knowledge, therefore, requires a definition of knowledge that accounts for bounds on an agent's computational power.

Recall that, while the need for such definitions of knowledge accounting for an agent's computational powers is acutely apparent in the context of cryptography, we have already seen the need for such definitions in Chapter 3, even in the absence of cryptography. In the sending or receiving omissions models, the tests for common knowledge used by an agent to determine whether a fact is common knowledge are easily-computable functions of the agent's local state. The same is typically true for most work in the literature using knowledge to analyze distributed computation. For this reason, using information-theoretic definitions of knowledge (definitions that do not take

into account agents' limited computational power) does not lead to trouble. In the generalized omissions model, however, the same tests for common knowledge are no longer easily computable. We therefore concluded in Chapter 3 that information-theoretic definitions do no capture all relevant aspects of simultaneous coordination in this model. A major challenge presented here, therefore, is to define knowledge in a way that accounts for bounds on agents' computational powers.

5.3.1 Knowledge and Probability

As we saw in Chapter 4, there are a number of meaningful definitions of probabilistic knowledge in the context of synchronous systems, systems such as the ones we consider here. Since the prover p seems to be the natural choice for the verifier's "opponent" in an interactive proof system, arguments in Chapter 4 imply that $\mathcal{P}^p$, the assignment that conditions on the joint knowledge of both the prover and the verifier, is the "right" assignment for the verifier to use. In this chapter, however, we will use the assignment $\mathcal{P}^{fut}$, the assignment that assigns to an agent and a point c the probability space of all points with c's global state.

The choice of this assignment is due to the fact that we will be interested in the truth of formulas of the form $K_i^\alpha\varphi$ at time 0 points. In Chapter 4, we noted that all of the assignments $\mathcal{P}^{fut}$, $\mathcal{P}^p$, $\mathcal{P}^{post}$, and even $\mathcal{P}^{prior}$ are equivalent at time 0; that is, the probability spaces they assign to a given agent at a given point are identical. This means that a formula $K_i^\alpha\varphi$ is true at time 0 with respect to one of these assignments iff it is true with respect to all of them. Consequently, from a semantic point of view, the exact choice of the assignment is irrelevant. From a computational point of view, however, $\mathcal{P}^{fut}$ has several advantages. First, the probability spaces assigned by $\mathcal{P}^{fut}$ are independent of the agent (they depend only on the current global state). Second, the probability space assigned to a point is uniquely determined by the distribution on the runs extending this point. Since interactive and zero knowledge proof systems are defined in terms a distribution on runs (that is, the distribution on the runs extending initial points), the definition of $\mathcal{P}^{fut}$ seems most closely related to the definition of such proof systems. Finally, the simple nature of $\mathcal{P}^{fut}$'s definition will simplify our analysis slightly.

With these observations in mind (that we can prove our results in terms of $\mathcal{P}^{fut}$ and know they will know in terms of any other assignment of interest,

and that $\mathcal{P}^{fut}$ simplifies our analysis), we fix $\mathcal{P}^{fut}$ as the probability assignment used in our analysis. Having fixed the assignment $\mathcal{P}^{fut}$, we can safely omit $\mathcal{P}^{fut}$ from the left side of the turnstyle '$\models$' in formulas involving probabilistic knowledge without introducing any ambiguity. Furthermore, since the operators Pr_i are identical for all agents p_i, we can omit the subscript i. We reiterate the point made in Section 5.2.1 concerning the formulas '$pr[\varphi] \geq \alpha$' and '$Pr[\varphi] \geq \alpha$': we write '$pr[\varphi] \geq \alpha$' when the underlying probability distribution is sequences of coin flips, and '$Pr[\varphi] \geq \alpha$' when this distribution is points or runs (and, in particular, when '$Pr[\varphi] \geq \alpha$' is meant to be interpreted as a formula in our language of knowledge and probability).

5.3.2 Knowledge and Computation

We now turn our attention to definitions of knowledge that account for an agent's limited computational power. Intuitively, we want to restrict an agent's knowledge to what it can compute. As Moses discusses in [Mos88], however, there is more than one way to do this. The motivation for our definition is that we want to use our definition of knowledge to construct and analyze protocols. The tests an agent uses to determine what it knows (and hence what actions to perform) in the course of a protocol are allowed to be virtually any function of the agent's local state. The *only* thing that restricts the tests an agent can perform is the agent's limited computational power. This is the fundamental intuition underlying our definition of *practical knowledge*, a definition of knowledge incorporating both probability and bounds on an agent's computational resources. The exact definition of practical knowledge is best motivated by way a sequence of intermediate definitions.

Resource-bounded knowledge

The definition of *resource-bounded knowledge* given in [Mos88] succinctly captures this intuition that it is the bounds on an agent's computational resources that restrict the tests the agent can perform, and hence what the agent can know. Loosely speaking, this definition says that a polynomial-time agent knows a fact only if there is a polynomial-time test the agent can use to determine that it knows this fact. This intuition can be generalized to any complexity class (see [Mos88]), and not just polynomial-time. However, since cryptography is typically concerned with what an agent can learn using

probabilistic tests running in time polynomial in some parameter determined by its local state (a parameter such as $|x|$, the length of the common input), the class BPP seems to be the complexity class of most relevance to cryptography. We therefore restrict our attention to knowledge with respect to the class BPP.

The notion of a BPP test an agent can use to determine whether it knows a fact φ can be made precise as follows. Given a system R, a probabilistic algorithm M is said to be a *BPP test for $K_q\varphi$ in R* if, for all points (r, m) of R,

1. M's input is q's local state $r_q(m)$,

2. M runs in time polynomial in $|x|$, where x is the common input recorded in $r_q(m)$,

3. M accepts with probability at least 2/3 if $(r, m) \models K_q\varphi$, and rejects with probability at least 2/3 if $(r, m) \not\models K_q\varphi$.[6]

This definition essentially says that the language of local states $r_q(m)$ satisfying $(r, m) \models K_q\varphi$ is in BPP, the only difference being that the BPP test is required to run in time polynomial in $|x|$ and not $|r_q(m)|$. We choose $|x|$ instead of $|r_q(m)|$ because it seems to be the preferred parameter in the context of interactive proofs. Interactive protocols (P, V) and simulating Turing machines M_{V^*}, for example, are both required to run in time polynomial in $|x|$, and not, say, in $|x|$, $|s|$, and $|t|$. In all interactive proofs we are aware of, however, the size $|r_v(m)|$ of the verifier's local state is polynomial in $|x|$.

We can now make precise the intuition that an agent knows a fact only if it can compute that it knows this fact. Given a system R, an agent q is said to *BPP-know* φ at a point c of R, denoted by $c \models K_q^{\mathrm{BPP}}\varphi$, if

1. $(r, m) \models K_q\varphi$, and

2. there is a BPP test for $K_q\varphi$ in R.

[6]The probability, of course, is being taken over M's coin flips. We note that there is nothing special about the value 2/3. We can use any value bounded above and away from 1/2. In fact, it is easy to replace any such value with $1 - 2^{-|x|}$ by using the standard trick of running the original test M many times to estimate the probability with which M accepts or rejects.

Thus, a processor BPP-knows φ if it knows φ and there is a BPP test it can use to compute that it knows φ.

To get a better feeling for how this definition behaves, consider a system in which an agent's local state includes two integer-valued variables m and n (the value of these variables might be determined by the contents of the input tape, for example), and suppose that for every pair of integers i_m and i_n there is a run of the system in which the values of m and n are i_m and i_n, respectively. Consider a point c at which $m \equiv (n-1)^2 \pmod{n}$. Since it is very easy for an agent to check that $m \equiv (n-1)^2 \pmod{n}$, it is clear that the agent BPP-knows the fact ψ that '$m \equiv (n-1)^2 \pmod{n}$' at the point c. Notice that if $m \equiv (n-1)^2 \pmod{n}$, then m is a quadratic residue modulo n (that is, a square modulo n). Since the agent BPP-knows that $m \equiv (n-1)^2 \pmod{n}$ at c, it is natural to assume that the agent must also BPP-know the fact φ that 'm is a quadratic residue modulo n' at c. But recall that in order for the agent to BPP-know the fact 'm is a quadratic residue modulo n' at a point, there must be a BPP test that determines whether m is a quadratic residue for arbitrary m and n; and assuming quadratic residuosity is hard, this is impossible. It follows that the agent does *not* BPP-know the fact 'm is a quadratic residue modulo n' at c after all. Notice that the agent BPP-knows the fact $m \equiv (n-1)^2 \pmod{n}$ at the point c, and since the implication "if $m \equiv (n-1)^2 \pmod{n}$, then m is a quadratic residue modulo n" is a tautology, the agent clearly BPP-knows this fact as well (the simple test that always accepts is a BPP-test for this fact). Consequently, this example shows that, unlike the information theoretic definition of knowledge, it is possible for an agent to BPP-know both facts ψ and $\psi \supset \varphi$ without BPP-knowing the fact φ. The agent *does* know $\psi \wedge \varphi$, but it need not know φ itself. In this sense, an agent no longer knows all consequences of its knowledge (that is, everything that logically follows from the information recorded in its local state). This is a result of the fact that this definition restricts an agent's knowledge to what it can compute. The reader is referred to [Mos88] for an interesting discussion of this and other properties of this definition.

A notion of learning

The definition of BPP knowledge restricts an agent's knowledge to what it can compute by requiring the existence of a test the agent can use at all points of a system to compute whether it knows a given fact. In this sense,

BPP knowledge captures what an agent can compute on its own. Sometimes, however, it is possible for an agent to obtain some extra information (possibly from another agent in the system), and with this extra information the agent is able to learn things it couldn't have computed on its own. This informal notion of "learning" is of great importance to cryptography (and, in particular, to zero knowledge proof systems). Unfortunately, it does not seem possible to capture this notion of learning directly in terms of resource-bounded knowledge.

To understand this situation more clearly, consider again the system in which an agent's local state contains the two integer-valued variables m and n, and consider again the fact φ that 'm is a quadratic residue modulo n.' As we have seen, it is impossible for an agent to BPP-know φ since there is no BPP test to determine whether m is a quadratic residue modulo n for arbitrary m and n. There are, however, situations in which it does seem to make sense to say that an agent knows φ. One example is the special case in which $m \equiv (n-1)^2 \pmod{n}$. A more interesting situation is one in which an agent somehow obtains the factorization of n, and hence the agent is easily able to compute whether φ holds. There are a number of ways in which the agent might obtain this factorization. The agent might find the factorization in one of the messages it has received from other agents in the system (e.g., from the prover in an interactive proof system); or, more generally, it might be able to deduce the factorization from the contents of these messages rather than finding the factorization explicitly contained in one of the messages. In either case it seems reasonable to say that, although the agent cannot always determine whether φ holds, in these cases it clearly can, and hence can be said to know φ. More generally, for any difficult to compute fact, once an agent has seen a proof of the fact, it no longer seems to make sense to say the agent does not know the fact (although it certainly did *not* know the fact *before* seeing the proof). Since an agent cannot BPP-know a fact like φ, however, this notion of learning cannot be captured directly in terms of resource-bounded knowledge.

Knowledge given facts

How, then, *can* one capture this notion of learning? We note that there are a number of ways of doing so, and at the end of this section we discuss several alternatives to the method we propose. Our approach, however, is a very

direct one. Recall the reason we felt resource-bounded knowledge could not capture this intuition: some agent may fortuitously obtain some information ψ, such as the factorization of an integer, that is enough for the agent to be able to determine that it knows a fact φ. Our idea is to define a notion of BPP knowledge of φ relative to a fact ψ. Roughly speaking, this means we have a BPP test M that correctly determines whether q knows φ when ψ is true, but is not necessarily correct when ψ is false. However, we do not want the results of this test to be completely arbitrary when ψ is false. In particular, we want to be able to trust this test whenever it says that q knows φ.

One way to capture this intuition is to make two requirements of the test M: the first is that M be a *sound* test for $K_q\varphi$, meaning that $K_q\varphi$ holds at a point if M accepts with high probability at that point; the second is that M be a *complete* test for $K_q\varphi$ at all points satisfying ψ, meaning that M will accept with high probability at such a point if $K_q\varphi$ holds at that point. These properties together guarantee that M is an accurate test for $K_q\varphi$ at points satisfying ψ; and soundness guarantees that, regardless of the truth of ψ, we can trust M when it says $K_q\varphi$ is true.

To make this precise, we proceed as follows. We say that a test M is a *sound* test for a fact ϑ at a point c if $c \models \neg\vartheta$ implies that M rejects at c with probability at least 2/3. We write $c \models \mathit{sound}(M, \vartheta)$ if M is a sound test for ϑ at c. Similarly, we say that M is a *complete* test for ϑ at c if $c \models \vartheta$ implies that M accepts at c with probability at least 2/3. We write $c \models \mathit{complete}(M, \vartheta)$ if M is a complete test for ϑ at c.

We capture the intuition that M is a good test for $K_q\varphi$ when ψ holds as follows. Given a system R, a probabilistic algorithm M is said to be a *BPP test for $K_q\varphi$ given ψ in R* if, for all points (r, m) of R,

1. M's input is q's local state $r_q(m)$,

2. M runs in time polynomial in $|x|$, where x is the common input recorded in $r_q(m)$,

3. M satisfies the following properties:

   (a) M is a *sound* test for $K_q\varphi$ on R: $R \models \mathit{sound}(M, K_q\varphi)$.

   (b) M is a *complete* test for $K_q\varphi$ given ψ: $R \models \psi \supset \mathit{complete}(M, K_q\varphi)$.

We remark that such a test is very similar to the solution of a *promise problem* as defined in [ESY84]. A promise problem (A, B) is a partial decision problem determined by two predicates, a promise A and a property B. A Turing machine N solves (A, B) if, for every x satisfying the promise $A(x)$, the machine N halts on input x and accepts on input x iff x satisfies the property $B(x)$. So N is a partial decision procedure for the language $L = \{x : B(x)\}$: it correctly determines whether $x \in L$ when the promise $A(x)$ is satisfied, but may behave arbitrarily when the promise $A(x)$ is not satisfied. Similarly, M is a decision procedure for $K_q\varphi$ when restricted to points satisfying the "promise" ψ, but may behave rather arbitrarily on the remaining points. The difference between a solution to a promise problem and such a test M is that M is required to be a sound test for $K_q\varphi$ even when ψ fails to hold.

We define knowledge of a fact φ given ψ as follows. Given a system R, we say that "*q knows φ given ψ*" at a point c, denoted by $c \models K_q^\psi\varphi$, iff

1. $c \models \psi$,
2. $c \models K_q\varphi$, and
3. there is a BPP test for $K_q\varphi$ given ψ in R.

The last two conditions, as in the definition of BPP knowledge, require that q actually knows φ and that there exists a feasible test M for $K_q\varphi$ that is sound in general, and complete given ψ. The first condition says knowledge given ψ holds only at points satisfying ψ. Intuitively, these points are the only points of interest since these are the only points where the promise ψ is true, the only points where q has learned the information sufficient for q to correctly determine whether it know φ. The fact that different tests M are allowed to behave differently at points failing to satisfy ψ is another reason we must require that $K_q^\psi\varphi$ hold only at points satisfying ψ: we want $K_q^\psi\varphi$ to be well-defined at all points, even points failing to satisfy ψ where the required behavior of our tests M is only loosely specified.

To understand the relationship between this definition of knowledge and resource-bounded knowledge, notice that if ψ is the fact **true**, then $K_q^\psi\varphi$ is equivalent to $K_q^{\mathrm{BPP}}\varphi$. In this sense, knowledge given a fact ψ is a direct generalization of resource-bounded knowledge. Furthermore, notice that if ψ is testable in BPP given only agent q's local state as input, then $K_q^\psi\varphi$ is

equivalent to $K_q^{\mathrm{BPP}}(\varphi \wedge \psi)$. In general, however, we do not restrict the facts ψ to be testable in BPP, and in this case it does not appear that knowledge given ψ can be captured directly in terms of BPP-knowledge.

To see how this notion of knowledge enables us to capture our intuition concerning learning, let us return to our initial example in which an agent q's local state includes two integer-valued variables m and n. Let φ be the fact that m is a quadratic residue modulo n, and let ψ be the fact that the factorization of n is explicitly given in the messages on q's communication tape. Let M be the test that accepts iff the factorization of n is explicitly given in the messages on q's communication tape and m is a quadratic residue modulo n. This test M for φ is clearly sound and clearly complete given ψ. Thus, when q learns from the factorization of n on its communication tape that φ is true, then q does indeed know φ given ψ.

We note, however that while the intuition motivating the definition of $K_q^{\psi}\varphi$ is that ψ is some additional information an agent might obtain that will enable it to determine whether it knows φ, the definition of $K_q^{\psi}\varphi$ is more general than this. Suppose, for example, that ψ is the fact that the prover in an interactive proof is the good prover. Intuitively, given that the verifier is talking to the good prover, the verifier knows $x \notin L$ when it rejects. The fact ψ, however, is a fact whose truth can never be determined given only the verifier's local state, and hence does not represent some information the verifier might somehow be able to learn, and therefore determine that it knows $x \notin L$. In this case, the right way to view ψ is not as a fact the verifier can learn, but as a condition or "promise" whose truth guarantees that the verifier's test M accurately determines whether it knows φ.

Finally, because the behavior of a test M is relatively unrestricted when the condition ψ is false, and because an agent may not be able to determine whether ψ is true or false, an important question is how an agent q is to interpret the result of running the test M. What meaning should q assign to the probability with which M accepts? Notice that M can accept either with probability less than 1/3 or with probability greater than 1/3 (and, in particular, with probability greater than 2/3). In the latter case, M's soundness guarantees to q that $K_q\varphi$ must hold, since M would accept with probability less than 1/3 if $K_q\varphi$ did not hold. On the other hand, q's ability to assign meaning to M's accepting with probability less that 1/3 depends on q's ability to determine whether ψ is true. If it can determine that ψ is true, then it is guaranteed that $\neg K_q\varphi$ holds. Otherwise, the test M gives q

no useful information about whether it does or does not know φ.

This discussion illustrates the asymmetry of the definition of $K_q^\psi\varphi$. In particular, since the test M may say $K_q\varphi$ does not hold when in fact it does (this can happen at a point failing to satisfy ψ), the tests associated with $K_q^\psi\varphi$ feel more like tests for $K_q\varphi$ than they do tests for $\neg K_q\varphi$. It seems, however, that positive tests about knowledge tend to be more important that negative tests in the context of cryptography. In the case of zero knowledge, for example, our intuition does not say that the verifier does *not* know a fact φ at the end of a proof of $x \in L$, but rather that if the verifier *does* know φ at the end of a proof, then it *also* knows φ at the beginning. Notice that proving a polynomial-time agent does *not* know a fact (say a fact it knows in the information-theoretic sense) would probably involve proving something about issues involving P versus NP. On the other hand, proving positive statements about a polynomial-time agent's knowledge involves the construction of polynomial-time tests, which is typically a much more tractable task. This probably explains the prevalence of positive statements about knowledge in cryptography.

Practical Knowledge

The definition of practical knowledge itself, the ultimate objective of this section, is obtained as a result of the following observation: a probabilistic test that fails on a negligible portion of its inputs is typically considered to be just as good as one that never fails. Similarly, in the context of zero knowledge, the fact that the distributions of $(P(s), \underline{V}^*(t))(x)$ and $M_{V^*}(t, x)$ can be distinguished by a polynomial-time test with only negligible probability is considered to be just as good as if the two distributions cannot be distinguished at all. The soundness and completeness conditions required by the definition of knowledge given ψ, however, do not allow for the possibility that a given test M might fail to be sound or complete at a negligible fraction of the points where we want it to be sound or complete. It is natural to consider relaxing these conditions in some way. In order to do this, we must first determine how we are going to go about measuring the size of the set of points where the test M fails. Since the only distribution available during probabilistic computation is the distribution on runs induced by the coins tossed during the runs, it seems most natural to require that the test behaves correctly at all points of all but a negligible fraction of the runs.

Formally, let *init* be the fact holding only at points at the beginning of a run (that is, at time 0 points). Given a system R, we say that M is a *practically sound* test for φ if for all k there exists α such that

$$R \models init \supset Pr(\Box sound(M, K_q\varphi)) \geq 1 - \alpha\,|x|^{-k}.$$

Similarly, given a fact ψ, we say M is a *practically complete* test for $K_q\varphi$ given ψ if for all k there exists α such that

$$R \models init \supset Pr(\Box[\psi \supset complete(M, K_q\varphi)]) \geq 1 - \alpha\,|x|^{-k}.$$

Notice that, since we want to consider tests that behave correctly on all but a small fraction of the runs, we have used the antecedent *init* in the definition of practical soundness and practical completeness to ensure that the probability is being taken over the runs of the system. These definitions are equivalent to saying that for every initial global state of the system, the conditions $sound(M, K_q\varphi)$ and $\psi \supset complete(M, K_q\varphi)$ hold at all points of almost all runs extending this initial global state. That is, these conditions are statements about prior probabilities. We could have considered instead tests with the stronger property that they behave correctly at all but a small fraction of the points extending *any* given global state (by deleting the antecedent *init*). This latter notion can lead to dramatically different results (recall the analysis of the probabilistic coordinated attack problem given in Chapter 4), but does not seem appropriate for most computer science applications. In particular, it does not seem appropriate in the context of interactive proofs: at a point where the verifier has already accepted, it no longer makes sense to expected the verifier to reject with high probability, even when $x \notin L$.

We now define "*q practically knows φ given ψ*" at a point c, which we denote by $c \models \tilde{K}_q^{\psi}\varphi$, in precisely the same way as we defined "q knows φ given ψ," except that the soundness and completeness conditions are replaced by practical soundness and practical completeness. Formally, $c \models \tilde{K}_q^{\psi}\varphi$ iff

1. $c \models \psi$,
2. $c \models K_q\varphi$, and
3. there is a test M that is practically sound for $K_q\varphi$ and practically complete for $K_q\varphi$ given ψ.

The tilde in the notation $\tilde{K}_q^{\psi}\varphi$ is intended to denote the approximate nature of the tests M guaranteed by the definition of practical knowledge. To say that an agent practically knows φ given ψ, therefore, means that the agent knows φ and has a test that quite accurately determines whether it knows φ at points satisfying ψ, although on rare occasions (that is, in a negligible fraction of the runs) it may make mistakes.

Alternate definitions

As we have mentioned, there are several alternatives to the definition of practical knowledge. Before proceeding to show how practical knowledge can be used to analyze interactive and zero knowledge proof systems, we discuss several of these alternatives. The reader interested only in the application of practical knowledge to interactive and zero knowledge proof systems can safely skip ahead to the beginning of the next section.

Recall once again the intuition motivating the definition of practical knowledge: as a result of learning the fact ψ that $m \equiv (n-1)^2 \pmod{n}$, an agent can deduce that it knows the fact φ that m is a quadratic residue modulo n. Notice that in this case the fact ψ is actually a proof of the fact φ. In general, knowing a proof of a fact φ is equivalent to knowing a stronger fact ψ that implies φ. Thus, since ψ is presumably easy to verify and φ is not, instead of talking about knowing φ, we could talk about knowing ψ (and hence φ) instead. But this is not very satisfactory. Returning to our quadratic residuosity example, what interests us is whether the agent knows this fact φ that m is a quadratic residue modulo n, and not the particular proof of φ the agent knows. We want to be able to describe protocols in terms of knowledge, such as "*if q knows φ, then q should halt and accept.*" If all we can talk about are the various proofs ψ of φ, however, then we are forced to describe this protocol indirectly with "*if, for any proof ψ of φ, agent q knows ψ, then q should halt and accept.*" Such descriptions seem much less desirable than the first.

To avoid this problem, one might be tempted to define a notion of learning in which an agent learns φ at a point if at this point it BPP-knows a fact ψ that implies φ, implicitly existentially quantifying over all possible proofs ψ of φ. Unfortunately, this notion of learning is not very useful to a resource-bounded agent. It could be, for example, that at every point c the agent $\overline{\text{BPP}}$-knows a different fact ψ_c implying φ (and hence has "learned" φ everywhere)

and yet is unable to determine at a particular point which fact ψ_c it should test for in order to determine that it knows φ.

Another approach one might be tempted to take is to define a notion of knowing φ with respect to a particular test M where, informally, an agent knows φ with respect to M if using the test M the agent can determine that it knows φ. We remark that Fischer and Zuck define a similar notion of knowledge in [FZ87], but based on RP tests instead of BPP tests. Notice, however, that in some sense this idea is very similar to BPP-knowing a particular proof ψ of φ, since we can always take the proof ψ to be the fact that M accepts with high probability (and hence tells us that φ holds). This approach consequently shares the disadvantages discussed above. On the other hand, instead of being forced to quantify over all possible proofs ψ of φ when describing protocols as we did above, we are now forced to quantify over all proofs ψ *and* all tests M verifying such proofs, compounding our original complaint. Most important, however, we want to be able to specify and analyze protocols in terms of knowledge precisely because we want to be able to abstract away the particular tests being used when we think about computation. We note that the definition of resource-bounded knowledge already existentially quantifies over such tests (so these tests do not appear in the notation used), and we do not want to reintroduce them here.

The reader may still wonder about the asymmetry of our definition. Why do we require soundness at all points, but completeness only at points satisfying ψ? Notice that if we strengthen the definition to require both soundness and completeness at all points, then we have essentially returned to the definition of BPP knowledge. On the other hand, suppose we weaken the definition to require soundness only at points satisfying ψ. If ψ is easily testable, then such a notion of knowledge may be of interest. As we have mentioned, however, we want to be able to consider facts ψ that are not easily testable, and in this context this weakening of our definition becomes rather uninteresting. For in contrast to our definition, q's ability to assign *any* meaning to M's probability of acceptance would now depend on q's ability to determine whether ψ is true, which makes M of little use if testing for ψ is hard. We could instead have required completeness at all points and soundness only a points satisfying ψ, but this would change the flavor of M's behavior from being primarily a test for $\bar{K}_q\varphi$ to being a test for $\neg\bar{K}_q\varphi$, which (as we have said) seems less relevant in the context of cryptography.

Finally, we note that in an earlier version of this work [HMT88] we defined

knowledge with respect to sets of points A instead of defining knowledge with respect to facts ψ. Intuitively, the set A consisted of the points in the system (for example, the points satisfying some fact ψ_A) where an agent has obtained enough information to be able to determine whether it knows a fact φ. The primary disadvantage of this way of defining knowledge is that the logic of knowledge used to analyze a system is no longer independent of the system being analyzed. It is no longer possible, for example, to argue that since the formula $\varphi' \supset K_q^{\psi}\varphi$ is valid in one system, it is valid in a second. Instead we must argue that since a formula like $\varphi' \supset K_q^{A}\varphi$ is valid in one system, a formula like $\varphi' \supset K_q^{B}\varphi$ is valid in a second for some set B of points related to the set A in some way that must be explicitly specified. Introducing such sets of points into our logic results in losing the abstraction from the operational nature of the system being studied that motivated us to avoid defining knowledge with respect to particular tests M in the first place.

5.4 Knowledge and Interactive Proofs

We now return to the study of knowledge and interactive proof systems. Notice that the cryptographic definition of an interactive proof system really has nothing to do with knowledge or computational complexity. It is simply a statement about probability. It is not surprising, therefore, that we can immediately translate the statements of soundness and completeness in the definition of an interactive proof system directly into our language of probability. Recall that *init* is the fact holding only at points at the beginning of a run (that is, time 0 points), and let *accept* be the fact holding only at points at which the verifier has accepted.

Proposition 5.1: An interactive protocol (P, V) is an interactive proof system for a language L iff the following conditions are satisfied:

- *Completeness*: For every $k \geq 1$ there exists $\alpha \geq 1$ such that

$$P \times V \models \mathit{init} \supset Pr[x \in L \supset \Diamond \mathit{accept}] \geq 1 - \alpha\,|x|^{-k}\,.$$

- *Soundness*: For every $k \geq 1$ there exists $\alpha \geq 1$ such that

$$\mathcal{P} \times V \models \mathit{init} \supset Pr[\Diamond \mathit{accept} \supset x \in L] \geq 1 - \alpha\,|x|^{-k}\,.$$

The proof of Proposition 5.1 (and all other results in this chapter) can be found in Appendix 5.A. The constant α used above is necessary due to the fact that the probabilistic guarantees made by the definition of an interactive proof system hold only for sufficiently large x. Notice that if $1 - \alpha |x|^{-k}$ is negative, then $Pr(\varphi) \geq 1 - \alpha |x|^{-k}$ is equivalent to $Pr(\varphi) \geq 0$, which is valid for every fact φ. Consequently, by choosing α so that $1 - \alpha |x|^{-k} < 0$ for insufficiently large x we obtain a formula holding for all x, and hence valid at all points of the system. While this constant α does not appear in the formal definition of an interactive proof system, an equivalent definition of interactive proof systems can be formulated making use of such constants just as we do in Proposition 5.1.

According to Proposition 5.1, a formula such as $Pr[x \in L \supset \Diamond accept] \geq 1 - \alpha |x|^{-k}$ holds at time 0 but not necessarily at later points. After the verifier has rejected, for example, it is clearly not the case that with high probability the verifier will eventually accept. In general, even before the verifier has actually decided to accept or reject, a particularly bad sequence of coin flips can significantly lower the verifier's chances of eventually accepting. Consequently, the antecedent *init* is crucial in the formulas above. Intuitively, this is due to the fact that the verifier's probability space is changing with every step. Since we have chosen the assignment $\mathcal{P}^{fut}$ as the basis for our definition of probabilistic knowledge, an assignment associating with a point the set of points having the same global state, an agent's probability space decreases in size with every step. The same would often be true if we had chosen any other consistent assignment such as $\mathcal{P}^{post}$ or $\mathcal{P}^{j}$.

Since the facts appearing in Proposition 5.1 are valid, all agents know these facts at all points. Furthermore, all agents know the fact *init* whenever it holds. Since from $K_q init$ and $K_q(init \supset \psi)$ we can deduce $K_q \psi$, we can immediately deduce the following corollary to Proposition 5.1.

Corollary 5.2: An interactive protocol (P, V) is an interactive proof system for a language L iff the following conditions are satisfied:

- *Completeness*: For every $k \geq 1$ there exists $\alpha \geq 1$ such that

$$P \times V \models init \supset E^{1-\alpha|x|^{-k}}(x \in L \supset \Diamond accept).$$

- *Soundness*: For every $k \geq 1$ there exists $\alpha \geq 1$ such that

$$\mathcal{P} \times V \models init \supset K_v^{1-\alpha|x|^{-k}}(\Diamond accept \supset x \in L).$$

This corollary says that (P, V) is complete if both the good prover and the good verifier know with high probability that if $x \in L$, then the good prover will convince the good verifier to accept; and (P, V) is sound if the good verifier knows with high probability that, no matter what protocol the prover is running, if the verifier accepts x then $x \in L$.

One important difference to notice between the two statements is that completeness is stated with respect to the system $P \times V$ consisting of the good prover and the good verifier, while soundness is statement with respect to $\mathcal{P} \times V$ consisting of arbitrary provers and the good verifier. In the a system $P^* \times V$, the prover P^* is fixed and hence the verifier knows which prover it is talking to. In the system $\mathcal{P} \times V$, however, the verifier may consider any prover possible, and hence cannot know the identity of the prover. In this way we are able to capture quite simply the intuition that the verifier can be confident that $x \in L$ whenever it accepts, regardless of which prover it has been talking to.

A second observation worth making here is that if (P, V) is sound, then it is actually the case that (in addition to the verifier) every prover also knows with high probability that $x \in L$ whenever the verifier accepts; that is, we could have replaced $K_v^{1-\alpha|x|^{-k}}$ by $E^{1-\alpha|x|^{-k}}$ in the statement of soundness above. We have chosen to formulate this statement in terms of the verifier's knowledge since our intuition says that soundness is intended to be primarily a guarantee to the verifier (just as zero knowledge is intended to be primarily a guarantee to the prover).

While Corollary 5.2 shows that it is possible to characterize interactive proof systems in terms of knowledge and probability, this characterization is a reformulation of the original cryptographic definition in terms of very similar concepts. It does not significantly clarify our intuition concerning interactive proof systems, other than making explicit this distinction between what is intended to be a guarantee to the prover and what is a guarantee to the verifier. It does not capture, for example, the intuition that at the end of an interactive proof of $x \in L$ with the good prover, the good verifier knows that $x \in L$ despite its limited computational power.

In what way can the verifier be said to know whether $x \in L$ at the end of a proof of $x \in L$? If our intuition is correct, the verifier knows $x \in L$ whenever it accepts. Consider the test M that takes as its input the verifier's local state and accepts at a point if the verifier has accepted at that point and rejects otherwise. Loosely speaking, the soundness condition for

an interactive proof implies that M will not accept when $x \notin L$, and the completeness condition implies that M will accept when $x \in L$ if M is run at the end of a proof with the good prover. Let us denote by ***halted*** the fact holding at a point iff at that point the verifier has either accepted, rejected, or otherwise halted. We refer to a point satisfying ***halted*** as a ***final point***. Let us denote by '*p running P*' the fact holding at a point iff at that point the prover is following the protocol P. Let ψ be the fact ***halted*** $\wedge$ '*p running P*'. Intuitively, we would like to say that the good verifier knows $x \in L$ given ψ at the end of a proof of $x \in L$ with the good verifier. Of course, the test M is not a sound test for $x \in L$ since on rare occasions the verifier may incorrectly accept when $x \notin L$, and M is not complete given ψ for similar reasons. On the other hand, it is ***practically sound*** and is ***practically complete*** given ψ. As a consequence, we can prove the following.

Proposition 5.3: If (P, V) is an interactive proof system for L, then

$$\mathcal{P} \times V \models (x \in L \wedge \text{'}p \textit{ running } P\text{'}) \supset \Diamond \tilde{K}_v^{\psi}(x \in L),$$

where $\psi \stackrel{def}{=}$ ***halted*** $\wedge$ '*p running P*'.

In fact, we can essentially prove a converse of this proposition as well, which shows that we can characterize the notion of an interactive proof system using practical knowledge.

Proposition 5.4: If

$$\mathcal{P} \times V^* \models (x \in L \wedge \text{'}p \textit{ running } P\text{'}) \supset \Diamond \tilde{K}_v^{\psi}(x \in L),$$

where $\psi \stackrel{def}{=}$ ***halted*** $\wedge$ '*p running P*', then we can effectively modify V^* to obtain V such that (P, V) is an interactive proof system for L.

The protocol V is simply the protocol V^* at the end of which the verifier uses its test for practical knowledge of $x \in L$ to decide whether to accept or reject.

These results tell us that an interactive proof system for L is precisely one that guarantees that the verifier will practically know $x \in L$ at the end of a proof of $x \in L$ with the good prover, and will practically never be fooled (by *any* prover). We remark that, having reformulated the cryptographic

definition of an interactive proof system in terms of our logic of knowledge and probability (recall Proposition 5.1), the proof of this new characterization of interactive proof systems has been done entirely by reasoning about formulas in our logic of knowledge and probability. We consider this to be quite important, since one of the major reasons for studying cryptography in terms of knowledge is to be able to reason at a semantic level about cryptographic systems without delving into the (often complex) operational nature cryptographic definitions and computation.

5.5 Knowledge and Zero Knowledge

We now turn our attention to zero knowledge proof systems, and show how to capture the intuition that if the verifier knows a fact φ at the end of a zero knowledge proof of $x \in L$, then the verifier knows $x \in L \supset \varphi$ at the beginning of the proof as well. Since this intuition requires that φ be true at the beginning of a proof whenever it is true at the end of a proof, it must be a fact that depends only on the information contained in the initial state and cannot be a fact like "the proof is over." Recall that, given a system R, a fact φ is said to be a fact *about the initial state* if $(r, m) \models \varphi$ implies $(r', m') \models \varphi$ for all points (r', m') in R with $r(0) = r'(0)$. That is, φ is a fact about the initial state if the truth of φ at a point of a run depends only on the run's initial state. Restricting our attention to facts about the initial state is not much of a restriction in practice since we are typically concerned that the prover will leak some information about the common input x to the verifier, and any fact about x is in particular a fact about the initial state (since x is encoded in the initial state).

The following theorem captures the intuition mentioned above. Roughly speaking, it says that if $x \in L$ and the verifier has a nontrivial chance of learning a fact φ at the end of a proof of $x \in L$, then the verifier can already deduce φ from $x \in L$ on its own at the beginning of the proof without interacting with the prover. Consequently, provided $x \in L$, the only information that a prover leaks to the verifier in a zero knowledge proof of $x \in L$ are facts that follow from $x \in L$. In this sense, the verifier learns essentially nothing as a result of the proof other than the fact $x \in L$ the prover set out to prove. However, the proviso that $x \in L$ is crucial here. There is nothing in the definition of a zero knowledge proof to stop the

prover from leaking all sorts of information when $x \notin L$.

Theorem 5.5: Let (P, V) be a zero knowledge proof system for L, let V^* be an arbitrary verifier, and let φ be a fact about the initial state. For every fact ψ and constant $k \geq 1$ there is a fact ψ' and a constant $\alpha \geq 1$ such that

$$P \times V^* \models (x \in L \wedge init) \supset K_p^{1-\alpha|x|^{-k}}[\Diamond \tilde{K}_v^{\psi} \varphi \supset \tilde{K}_v^{\psi'}(x \in L \supset \varphi)].$$

The statement of this theorem is one of the major motivations for the definition of practical knowledge. We want to capture the idea that if the verifier is able to compute something on its own as a result of obtaining some extra information (represented by the fact ψ) from the prover during the course of a proof, then the verifier is already able to compute this on its own at the beginning of the proof. BPP-knowledge does not seem to let us capture this intuition. We note, however, that the same result holds when we replace practical knowledge given ψ by BPP-knowledge given ψ, but this strengthening of the hypothesis (that the verifier knows φ given ψ at the end of the proof) weakens the statement of the theorem. Furthermore, the characterization of interactive proof systems in terms of practical knowledge given by Propositions 5.3 and 5.4 in Section 5.3.2 indicates that practical knowledge is of greater relevance to interactive proof protocols. Loosely speaking, the fact ψ' represents the condition that the current point is an initial point with $x \in L$, and that from this initial point there is a nonnegligible chance that $\tilde{K}_v^{\psi}\varphi$ will hold at the end of the run. The test for $x \in L \supset \varphi$ that the verifier uses at such points essentially runs the simulating Turing machine repeatedly to generate local histories (since $x \in L$, this simulation is guaranteed to be quite accurate), and runs the test for φ at the end of each of these histories. Since this test will succeed at the end of a nonnegligible fraction of these histories, by generating enough of them the verifier is almost certain to generate one such history, at which point it can accept.

Stepping back and looking at the statement of Theorem 5.5, however, we see that the result is slightly unsatisfactory. The reason is that it is stated in terms of the system $P \times V^*$, and in this system the verifier's protocol V^* is fixed and hence known to the prover. In contrast, the intuition behind zero knowledge is that even though the prover does not know the identity of the verifier, the prover knows that the verifier learns nothing at the end of the proof other than $x \in L$. In other words, our intuition suggests that the statement of Theorem 5.5 should also hold in the system $P \times \mathcal{V}^{pp}$.

Unfortunately, we cannot prove such a result. Given a test N for $K_v\varphi$ at the end of a proof of $x \in L$ in the system $P \times V^*$, our proof of Theorem 5.5 constructed a test M for $K_v(x \in L \supset \varphi)$ at the beginning of the proof by repeatedly running M_{V^*} to generate runs of $P \times V^*$ and running the test N at the end of the generated run. In order to do the same thing in the system $P \times \mathcal{V}^{pp}$, because we require that our test M behave correctly at all points of the system, M must first be able to determine the identity of the simulating Turing machine M_{V^*} given the identity of the verifier's protocol V^*. But since the order of quantification in the definition of zero knowledge guarantees only that for every verifier V^* there is a Turing machine $M_{V^*}(t, x)$ approximating the distribution of $(P(s), V^*(t))(x)$, there is no guarantee that there is a uniform way of choosing M_{V^*}. This is a rather subtle point brought out by our framework.

Since the source of this trouble seems to be the nonuniformity of M_{V^*}, a natural solution is simply to require that the simulating Turing machine is indeed uniform in the verifier's protocol; that is, require that one Turing machine M using V^* as a subroutine can simulate the runs of (P, V^*) for every verifier protocol V^*. We remark that most known zero knowledge protocols already have this property. This property is captured by the notion of black-box zero knowledge. An interactive proof system (P, V) for L is said to be ***strongly black-box zero knowledge*** (cf. [Ore87]) if there is a probabilistic Turing machine M such that

1. $M(V^*, t, x)$ runs in expected time polynomial in $|x|$, and

2. the families

$$\{(P(s), \underline{V}^*(t))(x) : (x, s, t) \in Dom\} \text{ and } \{M_{V^*}(t, x) : (x, s, t) \in Dom\}$$

   are polynomially indistinguishable, where $(x, V^*, s, t) \in Dom$ iff $x \in L$, V^* is a possible verifier protocol, s is a possible input for P, and t is a possible input for V^*.

If (P, V) is a strongly black-box zero knowledge proof system for L, then we can prove the analogue of Theorem 5.5 (with virtually the same proof) in the system $P \times \mathcal{V}^{pp}$ instead of $P \times V^*$:

Theorem 5.6: Let (P, V) be a strongly black-box zero knowledge proof system for L, and let φ be a fact about the initial state. For every fact ψ and constant $k \geq 1$ there is a fact ψ' and a constant $\alpha \geq 1$ such that

$$P \times V^{pp} \models (x \in L \wedge init) \supset K_p^{1-\alpha|x|^{-k}}[\Diamond \tilde{K}_v^{\psi} \varphi \supset \tilde{K}_v^{\psi'}(x \in L \supset \varphi)].$$

Unfortunately, as the name suggests, the notion of strongly black-box zero knowledge is stronger than one might expect most protocols to satisfy. The problem is that in practice $M(V^*, t, x)$ runs V^* as a subroutine on input x. Even if M runs V^* only once, the running time of M is at least as great as the running time of V^*. Consequently, even if we restrict our attention to polynomial-time V^* as input to M, since the polynomial bound on the running time of V^* is different for every V^*, the running time of M will not be bounded by any single polynomial. Oren avoids this problem in his definition of black-box zero knowledge by charging only one time step for a call to V^*. Thus, he is essentially viewing M as an oracle machine (rather than a purely polynomial-time Turing machine). We could modify our definitions to allow for knowledge with respect to oracle machines, but a more natural solution is to modify the measure we use of a test's complexity. In particular, suppose we consider tests for facts that run at a point (r, m) in time polynomial in $|x|$, the running time of V^*, and the description of V^*, where r is a run with input x in which the verifier is running the protocol V^*. Then, defining a notion of practical knowledge with respect to such tests, the analogue of Theorem 5.5 follows with precisely the same proof. We note that all zero knowledge protocols we are aware of satisfy this notion of black-box zero knowledge.

5.6 Generation and Zero Knowledge

In the previous section we formalized the idea that the verifier in a zero knowledge proof learns essentially nothing but the fact the prover sets out to prove. This is not, however, the strongest notion of security one could hope for. It would also be desirable to show that, as a result of interacting with the prover, the verifier cannot *do* anything that it could not do before the interaction. As mentioned in the introduction, for example, there is a big difference between knowing an integer n is composite and being able to generate a factor of n.

We abstract the idea of the verifier being able to do something as *knowing how to generate a y such that $R(x,y)$*, where R is simply a binary relation. For example, if $R(x,y)$ holds precisely when y is a prime factor of a number x on the input tape, then being able to generate a y such that $R(x,y)$ means being able to find a prime factor of x. Notice that, as in the case of factoring, many natural relations R are testable in BPP given both x and y as input, even though generating a y satisfying $R(x,y)$ given only x as input may be intractable. The assumption that a relation R is testable in BPP, therefore, is generally not a severe restriction. Formally, a relation R is *testable in BPP* if there is a probabilistic algorithm running in time polynomial in $|x|$, accepting (x,y) with probability at least 2/3 if $R(x,y)$, and rejecting (x,y) with probability 2/3 if $\neg R(x,y)$.

Just as we have said that the verifier knows a fact φ if it has an algorithm to test for φ, we would like to say that the verifier knows how to generate a y satisfying $R(x,y)$ if it has an algorithm to generate such a y. When defining knowledge of facts, we have considered tests for facts φ that were sound and were correct given that a certain other fact ψ was true. Here, although there are no conditions analogous to soundness and completeness, we consider algorithms that do a "good job" of generating y's such that $R(x,y)$ at points satisfying ψ, but may not perform as well at other points. Given a system R, we say that a probabilistic algorithm M is a *generator for R given ψ* for an agent q if for every point (r,m) of R

1. M takes as input q's local state $r_q(m)$ at (r,m),

2. M runs in time polynomial in $|x|$, where x is the common input recorded in $r_q(m)$, and

3. if M outputs a string y then $R(x,y)$ holds, and if (r,m) satisfies ψ then M outputs such a string with probability at least 2/3.

This requirement that M never incorrectly outputs a string y failing to satisfy $R(x,y)$ is easy to enforce when R is testable in BPP.

Given a system R, we say that the verifier *knows how to generate a y satisfying $R(x,y)$ given ψ* at a point c, which we denote by $c \models G_v^{\psi}\, y.R(x,y)$, if

1. $c \models \psi$, and

2. there is a generator for R given ψ for v.

Before we continue, it is helpful to consider the relationship between this definition of knowing how to generate and the definition of knowing a fact. It is natural to suppose that knowing a fact can be characterized in terms of knowing how to generate. For example, suppose $\varphi(x)$ is a fact about x, and suppose R is the relation defined by $R(x,1)$ if $\varphi(x)$ is true and $R(x,0)$ if $\varphi(x)$ is false. Knowing how to generate a y such that $R(x,y)$ given ψ implies knowing $\varphi(x)$ given ψ. To see this, suppose N is a generator for R given ψ, and suppose M is the test for $\varphi(x)$ that accepts at a point iff N outputs 1, and rejects otherwise. M must be sound, since N never outputs an incorrect string y, and hence N outputs 0 if it outputs anything at all when $\varphi(x)$ is false. On the other hand, M must be complete given ψ, since at points satisfying ψ the generator N outputs 1 with probability 2/3 when $\varphi(x)$ is true, and hence M accepts with probability 2/3. But what about the other direction? Does knowing $\varphi(x)$ given ψ imply knowing how to generate a y satisfying $R(x,y)$ given ψ? If R is testable in BPP, then an agent actually knows how to generate a y satisfying $R(x,y)$ given the fact **true**, and hence also given the fact ψ. But if R is testable in BPP, then so is $\varphi(x)$ and hence so is membership in the language L. For more interesting languages L, namely languages not contained in BPP, it seems possible that an agent can know $\varphi(x)$ given ψ without knowing how to generate a y satisfying $R(x,y)$ given ψ. In other words, knowing the existence of a proof that $x \in L$ seems to be different from knowing how to generate a proof that $x \in L$. Intuitively, the reason for this is that a BPP test M for knowledge of $\varphi(x)$ given ψ is allowed to make mistakes, whereas a generator N for $R(x,y)$ given ψ is not. For example, given such a test M, suppose we try to construct such a test N in the obvious way by having N output 1 if M accepts and 0 otherwise. M can reject outright at any point not satisfying ψ regardless of whether $\varphi(x)$ is true, and at such points N incorrectly outputs 0. We note, however, knowing how to generate is most interesting in contexts other than language membership, contexts such as factorization sketched above, and in these contexts the relations R *are* testable in BPP.

In any case, we can prove the following analogue to Theorem 5.5 (with virtually the same proof):

Theorem 5.7: Let (P,V) be a zero knowledge proof system for L, let V^* be an arbitrary verifier, and let $R(x,y)$ be a relation testable in BPP. For

every fact ψ and constant $k \geq 1$ there is a fact ψ' and a constant $\alpha \geq 1$ such that

$$P \times V^* \models (x \in L \wedge init) \supset K_p^{1-\alpha|x|^{-k}}[\Diamond G_v^{\psi}\, y.R(x,y) \supset G_v^{\psi'}\, y.R(x,y)].$$

Intuitively, this statement says that if the verifier has a nonnegligible chance of being able to generate a y satisfying $R(x,y)$ by talking to the prover, then the verifier can generate such a y on its own. We note that this theorem has a number of natural extensions. One simple extension is from generating y's satisfying relations $R(x,y)$ to generating y's satisfying facts φ about the verifier's entire initial state. Another simple extension, along the lines of practical knowledge, is a notion of practically knowing how to generate, denoted by $\tilde{G}_q^{\psi} y.R(x,y)$, where the algorithm may on a small fraction of the points satisfying ψ fail to generate y such that $R(x,y)$. A final extension, using black-box zero knowledge, allows us to prove an analogous result in the system $P \times \mathcal{V}^{PP}$.

We note that the ability to test the relation R in BPP is crucial to the proof of Theorem 5.7. Recall that in the proof of Theorem 5.5 the verifier tests for the fact φ by repeatedly generating runs and testing for φ at the end of each run. Since this test for φ is sound, the verifier can accept as soon as this test for φ accepts. Here, however, since there is no notion analogous to soundness, the verifier has no way of knowing which of the many y's it generates satisfies $R(x,y)$ and should be output unless the relation $R(x,y)$ can be tested in BPP. As we have said, however, most relations R of interest are testable in BPP.

Finally, we note that our definition of knowing how to generate given ψ is somewhat similar to the definition of probabilistic relative knowledge defined in [FZ87]. The only significant difference is that they define knowing how to generate relative to a particular Turning machine M, whereas we define knowing how to generate relative to a fact ψ. Roughly speaking, taking ψ_M to be the fact true at points where the test M outputs with probability 2/3 a y satisfying $R(x,y)$, knowing how to generate relative to M and knowing how to generate given ψ_M coincide. The natural generalization of our definition to practically knowing how to generate (where we allow the generator to make mistakes, but only on a negligible fraction of the runs) differs in subtle ways, however, from the generalization given by Fischer and Zuck.

5.7 Resource-bounded provers

In an interactive proof system as defined in [GMR89], the prover is assumed to be infinitely powerful. In practice, however, a prover is not infinitely powerful and may have no more computational power than the verifier. Fortunately, a probabilistic, polynomial-time prover with some "secret information" on its work tape is able to carry out many of the interesting interactive protocols. In the case of the graph isomorphism protocol from [GMW86] discussed in the introduction, for example, this secret information is an isomorphism between the graphs on the input tape. Since the context of such weak (polynomial-time) provers is actually the context of most practical interest, the type of security afforded by zero knowledge protocols in this context is an important question, and the subject of our final section.

In order to study zero knowledge proofs in this context, we define the notion of a weak interactive proof system, a direct modification of the definition of an interactive proof system for L. We define a *weak interactive protocol* to be an interactive protocol (P, V) where both P and V run in probabilistic, polynomial-time. We define a *weak interactive proof system* (P, V) *for a language* L just as we defined an interactive proof system for L except that we require (P, V) to be a weak interactive protocol and we restrict the quantification of P^* in the soundness condition to be only over probabilistic, polynomial-time machines, rather than over all machines. As the following lemma shows, however, weak interactive proofs of language membership are not very interesting.

Lemma 5.8: There is a weak interactive proof system for L iff L is in BPP.

Thus, an interesting weak interactive proof cannot be simply a proof of language membership; it must reveal something about the prover's local state, and hence must reveal something about the prover's knowledge since the prover's knowledge is determined by its local state. Consider again the zero knowledge proof of graph isomorphism from [GMW86] discussed in the introduction, or the zero knowledge proof of three-colorability also given in [GMW86]. Both proofs can be carried out by a weak prover with the appropriate information on its work tape, and in both cases the verifier obtains some information about the prover's knowledge as well as about language membership. In the case of graph isomorphism, the verifier learns that with

high probability the prover can generate an isomorphism between the graphs in question. Similarly, in the case of three-colorability, the verifier learns that with high probability the prover can generate a three coloring of the graph in question. It is well-known (see [HM84, MDH86]) that information about the prover's knowledge can dramatically affect the verifier's knowledge about the world. For example, in the case of three-colorability, information about the prover's knowledge may indicate to the verifier that the prover has with high probability communicated with the entity that generated the three-colorable graph.

In order to study proofs of the prover's knowledge, we extend the definition of a weak interactive proof of language membership to that of a weak interactive proof about the prover's initial state, where a fact is a fact about the prover's initial state if it depends only on the prover's initial state as defined in Chapter 2. Since the prover's initial state is determined by its protocol P^*, its initial work tape s, and the common input x, it is convenient to think of these components as parameters and denote facts about the prover's initial state by $\varphi(P^*, x, s)$. The definition of a weak interactive proof of $\varphi(P^*, x, s)$ is obtained simply by replacing all occurrences of $x \in L$ by $\varphi(P^*, x, s)$ in the definition of a weak interactive proof of language membership. Formally, we define a ***weak interactive proof system*** for a fact φ about the prover's initial state to be a weak interactive protocol (P, V) such that

- *Completeness*: For every k and sufficiently large x, and for every s and t, if $\varphi(P, x, s)$ then

$$pr[(P(s), V(t))(x) \text{ accepts}] \geq 1 - |x|^{-k}.$$

- *Soundness*: For every k and sufficiently large x, for every probabilistic, polynomial-time P^*, and for every s and t, if $\neg\varphi(P^*, x, s)$ then

$$pr[(P^*(s), V(t))(x) \text{ accepts}] \leq |x|^{-k}.$$

The reader may wonder why we consider weak interactive proofs of facts about the prover's initial state that depend on the prover's protocol as well as its work tape. To see why, suppose $\varphi(x, s)$ is a fact about the prover's work tape and the common input; that is, the truth of $\varphi(x, s)$ depends only on the prover's work tape s and the common input x (and not on the prover's protocol). Let us define $dom(\varphi)$ to be the set $\{x : \varphi(x, s) \text{ for some } s\}$.

Lemma 5.9: A weak interactive protocol (P, V) is a weak interactive proof system for a fact φ about the prover's work tape and the common input iff

1. for all sufficiently large x and for all s, we have $\varphi(x, s)$ iff $x \in dom(\varphi)$; and

2. $dom(\varphi)$ is in BPP.

This lemma says that if there is a weak interactive proof of a fact R about the prover's work tape and the common input, then R is essentially uninteresting. In particular, with the exception of a few small values of x, $\varphi(x, s)$ holds for all s whenever it holds for any s. Consequently, R is essentially determined by $dom(\varphi)$. Since $dom(\varphi)$ is in BPP, the prover can determine whether R holds (for sufficiently large x) without even interacting with the prover. Consequently, a fact R about the prover's initial state having only nontrivial weak interactive proofs must necessarily be a fact depending on the prover's protocol, and hence on the prover's entire initial state. Since the prover's knowledge is determined by its local state, such a weak interactive proof may be viewed as a proof of the prover's knowledge. In fact, we note that even in the context of infinitely powerful provers an interactive proof of $x \in L$ is not just a proof of $x \in L$ but a proof the prover knows $x \in L$ (i.e., a proof of the prover's knowledge). The fact that all interesting interactive proofs must be proofs of the prover's knowledge is obscured in the context of infinitely powerful provers since $x \in L$ holds iff the prover knows $x \in L$. In the context of weak prover, however, these facts are not equivalent.

We have defined a natural notion of interactive proof in the context of weak provers, and we have shown that the only nontrivial interactive proofs in this context are proofs about the prover's knowledge. While our definition is a direct modification of the definition in the case of strong provers, it is not initially clear that our definition is the most appropriate (or at all appropriate) in the context of weak provers, it is possible that our results are merely artifacts of our definition. As evidence supporting our definition, we now show that, under certain natural conditions, both interactive proof systems involving weak provers that have appeared in the literature [FFS87, TW87] are instances of weak interactive proofs. Not surprisingly, in light of our previous results, these proof systems concern proofs of the prover's knowledge.

In [TW87] we find the following definition (modified slightly for the sake of consistency with the rest of this chapter). Given a binary relation R, a weak interactive protocol (P, V) is said to be *an interactive proof that the prover can generate some y satisfying $R(x, y)$* if the following conditions are satisfied:

- *Completeness*: For every $k \geq 1$ and sufficiently large x and for every s and t, if $R(x, s)$, then

$$pr[(P(s), V(t))(x) \text{ accepts}] \geq 1 - |x|^{-k}.$$

- *Soundness*: For every probabilistic, polynomial-time P^* there is a probabilistic Turing machine M_{P^*} running in time polynomial in $|x|$ such that for every $k \geq 1$ and sufficiently large x and for all s and t,

$$pr[V \text{ accepts at } (r, m) \supset R(x, M_{P^*}(r_p(m)))] \geq 1 - |x|^{-k}$$

  where the probability is taken over the runs r of $(P^*(s), V(t))(x)$ and the coin flips of M_{P^*}.[7]

While we would like to show that every interactive proof that the prover can generate some y satisfying $R(x, y)$ is a weak interactive proof, this is not quite true. To see this, notice that the definition of a weak interactive proof requires that the probability with which $(P(s), V(t))(x)$ accepts is very close to 0 when $R(x, s)$ fails to hold, while an interactive proof of [TW87] allows the probability with which $(P(s), V(t))(x)$ accepts to be arbitrary as long as the prover P is able to generate a y satisfying $R(x, y)$. For example, if P is able to generate a y satisfying $R(x, y)$ with probability 1 at all points of the system, then $pr[V \text{ accepts at } (r, m) \supset R(x, M_{P^*}(r_p(m)))] = 1$ regardless of the probability with which the verifier accepts. We will prove below, however, that the following is a necessary and sufficient condition for an interactive proof of [TW87] to be a weak interactive proof:

- *Correctness*: For every $k \geq 1$ and sufficiently large x and for every s and t, if $R(x, s)$ does not hold, then $pr[(P(s), V(t))(x) \text{ accepts}] \leq |x|^{-k}$.

[7] We note that the soundness condition in [TW87] actually quantifies over *all* Turing machines P^* and not just over polynomial-time P^*. This is done for technical complexity-theoretic reasons. Since, however, the motivation for considering weak provers is that in practice all agents are restricted to polynomial-time, our restriction does not seem unnatural.

Intuitively, the good prover "tries" to convince the verifier to accept only when $R(x,s)$ holds. It is easy to show that, given an interactive proof of [TW87], this interactive proof can be modified to satisfy the correctness condition iff $R(x,y)$ is testable in BPP: the modification simply has the prover run the BPP test in order to determine whether is should attempt to convince the verifier to accept. Since this seems to be the most relevant context in practice (the relations used in the examples in [TW87] are testable in BPP, and [FFS87] explicitly restricts to *deterministic* polynomial-time relations[8]), this seems to imply that the correctness condition is a natural restriction. In the following proposition we show that (P,V) is an interactive proof of [TW87] for a relation R satisfying the correctness condition iff it is a weak interactive proof of the fact φ_R defined by

$$\begin{aligned}\varphi_R(P^*,x,s) \stackrel{def}{=} \; & (P^* = P \wedge R(x,s)) \vee \\ & (P^* \neq P \wedge \text{'}\textit{the soundness condition holds for } P^*\text{'})\end{aligned}$$

Note that φ_R depends on the prover's protocol as well as the work tape, and is a fact about the prover's initial state. Of course, φ_R is not necessarily testable in BPP.

Proposition 5.10: (P,V) is an interactive proof satisfying the correctness condition that the prover can generate a y such that $R(x,y)$ iff (P,V) is a weak interactive proof system for φ_R.

We can show, in addition, that the proof systems of [FFS87] satisfying the correctness condition above are also instances of a weak interactive proof system. The following is an interpretation of the quite informal definition of an interactive proof given in [FFS87]:

- *Completeness*: For every $k \geq 1$ and sufficiently large x and for every s and t, if $R(x,s)$, then

$$pr[(P(s),V(t))(x) \text{ accepts}] \geq 1 - |x|^{-k}.$$

[8] [Slo89] shows that certain anomalies in the definition of an interactive proof in [FFS87] disappear when the deterministic restriction is removed.

- *Soundness*: For every $k \geq 1$ there exists a probabilistic Turing machine M_k such that for every P^* and $\ell \geq 1$ and sufficiently large x, and all s and t,

$$pr[(P^*, V) \text{ accepts}] \geq |x|^{-k}$$

implies

$$pr[R(x, M_k(P^*, x))] \geq 1 - |x|^{-\ell}.$$

Here M_k is given the "code" for P^* and is allowed to run in time polynomial in x, the running time of P^*, and the length of the "code" for P^*.

It is not hard to show that such an interactive proof is also an interactive proof of a fact similar to φ_R. We leave the proof to the reader.

In light of the preceding propositions, our definition of a weak interactive proof system seems to be an appropriate definition; it can at least capture the definitions of other proof systems defined in the context of polynomial-time provers. We now turn to the study of the security afforded by such protocols. Our definition of a weak interactive proof is a direct modification of the definition of an interactive proof of language membership. We can also directly modify the definition of a zero knowledge proof of language membership to obtain a definition of a zero knowledge weak interactive proof: a weak interactive proof (P, V) is said to be *zero knowledge* if for every V^* there exists a Turing machine M_{V^*} such that the families

$$\{(P(s), V^*(t))(x) : (P, s, V^*, t, x) \in Dom\}$$

and

$$\{M_{V^*}(t, x) : (P, s, V^*, t, x) \in Dom\}$$

are polynomially indistinguishable, where $(P, s, V^*, t, x) \in Dom$ iff V^* is a possible verifier protocol, s and t are possible work tapes, and $\varphi(P, s, x)$.

Not surprisingly, analogues of all our previous results for interactive proofs hold in the case of weak interactive proofs, with essentially the same proofs. Rather than restating all the results here, we focus on one of them, the analogue of Proposition 5.1. If φ is a fact about the prover's initial state, then we say $(r, m) \models \varphi$ if $\varphi(P^*, x, s)$, where P^* is the protocol that p is running in r, x is the common input in the initial state $r(0)$, and s is the contents of p's work tape in $r(0)$.

Proposition 5.11: A weak interactive protocol (P, V) is a weak interactive proof system for a fact φ about the prover's initial state iff the following conditions are satisfied:

- *Completeness*: For every k there exists α such that

$$P \times V \models init \supset Pr[\varphi \supset \Diamond accept] \geq 1 - \alpha\,|x|^{-k}$$

- *Soundness*: For every k there exists α such that

$$\mathcal{P}^{pp} \times V \models init \supset Pr[\Diamond accept \supset \varphi] \geq 1 - \alpha\,|x|^{-k}.$$

Thus, we have replaced the occurrences of $x \in L$ in Proposition 5.1 by φ, and used $\mathcal{P}^{pp}$ rather than $\mathcal{P}$ in the soundness condition since we are restricting to weak provers.

At this point, we can make an interesting observation about the definition of interactive proof systems. Notice that in our soundness condition, the meaning of "sufficiently large x" (that is, the value of N_k) depends only on the value of k and not on the choice of P^*. In early versions of the definition of an interactive proof given in [GMR89], it is not clear whether the dependence is on k alone or on both k and P^*. But as Shafi Goldwasser pointed out to us, in the case of infinitely powerful provers, it doesn't matter what choice we make. More formally, in the context of language recognition, an interactive proof system (P, V) is sound with respect to one choice iff it is sound with respect to the other. The proof of this observation is a consequence of Feldman's proof technique for proving that it is sufficient to assume the prover's computational powers are limited to PSPACE [Fel]: we can construct a cheating PSPACE prover that, at any point during a conversation with the verifier V, can try all possible answers to the verifier's latest question, compute which answer will cause the verifier to accept with the greatest probability, and send this answer to the verifier.

In the case of weak provers, however, the order of quantification in the statement of soundness is important. In particular, if we had stated our soundness condition so that the choice of "sufficiently large x" might depend on the protocol P^*, all we would be able to prove is that for every k and every protocol P^*, there exists α such that

$$P^* \times V \models init \supset Pr[\Diamond accept \supset \varphi] \geq 1 - \alpha\,|x|^{-k}.$$

Instead, we can prove that for every k there exists an α such that

$$\mathcal{P}^{pp} \times V \models init \supset Pr[\Diamond accept \supset \varphi] \geq 1 - \alpha\,|x|^{-k}.$$

The first statement says that, for every prover, as long as the verifier knows the identity of the prover, φ is true whenever the verifier accepts. The second statement, on the other hand, says that no matter who the prover is, φ is true whenever the verifier accepts, which is clearly the desired statement. We remark that the weak interactive protocols resulting from the interactive proofs and zero knowledge proofs we are aware of satisfy the stronger notion of soundness we have used in our definition, and the revised definition of an interactive proof appearing in [GMR89] is consistent with the definition we use.

In addition to proving the analogues of results holding in the context of strong provers, we can reason about the interactive proofs of [FF87, TW87] directly in terms of the notions of knowledge and generation we have defined in previous sections. For example, we can characterize proofs that the prover can generate some y satisfying $R(x,y)$ just as we characterized interactive proofs, in the case that $R(x,y)$ is testable in BPP.

Proposition 5.12: Given a relation $R(x,y)$ testable in BPP, a weak interactive protocol (P,V) is a weak interactive proof that the prover can generate some y satisfying $R(x,y)$ iff the following conditions are satisfied:

- *Completeness*: For every k there exists α such that

$$P \times V \models init \supset Pr[R(x,s) \supset \Diamond accept] \geq 1 - \alpha\,|x|^{-k}$$

- *Soundness*: For every probabilistic, polynomial-time P^*,

$$P^* \times V \models accept \supset \tilde{G}_p^{\psi} y.R(x,y)$$

  where ψ is the fact *halted* that the verifier has halted.

Notice that in the soundness condition, we have $accept \supset \tilde{G}_p^{\psi} y.R(x,y)$ rather than $\Diamond accept \supset \tilde{G}_p^{\psi} y.R(x,y)$. The first condition says that the prover can generate some y such that $R(x,y)$ at the point when the verifier accepts, as required by [TW87], and not at the initial point as would be the case with the second clause. This is one of the differences between the definitions of

[TW87] and [FFS87]. A second difference between the two definitions is that the soundness condition of [FFS87] is such that we can state the soundness condition above in terms of the system $\mathcal{P}^{pp} \times \mathcal{V}$ instead of $P^* \times V$. We remark that because the machine $M(P^*, x)$ guaranteed by the definition of an interactive proof in [FFS87] runs in time polynomial in $|x|$, the running time of P^*, and the length of the encoding of P^*, we must modify the definition of $\tilde{G}_p^{\psi} y.R(x,y)$ to say that the generating Turing machine also runs in these parameters in order to reason about this definition of an interactive proof. This modification is the same modification needed to reason about notions of zero knowledge other than strong black-box zero knowledge.

5.8 An Application

In preceding sections we have characterized interactive proof systems in terms of knowledge. As an example of how to reason about interactive proof systems in terms of knowledge, we show how to prove the familiar result that the sequential composition of an interactive proof of $x \in L$ followed by an interactive proof of $x' \in L'$ is an interactive proof of $(x, x') \in L \times L'$.

For expository simplicity, we have been studying interactive protocols (P, V) in isolation. However, as shown by the coin flipping example in the introduction motivating interest in zero knowledge in the first place, interactive protocols are not used in isolation. They are intended to be used as subroutines or building blocks in the construction of other protocols. Providing a general definition of what it means for one protocol to be used as a subroutine in another protocol is a difficult problem. It is not too difficult, however, to define the sequential composition of two protocols.

Loosely speaking, if P and Q are two protocols, their sequential composition $P;Q$ should correspond to first running the protocol P until it halts (if ever) and then running the protocol Q. Recall that a protocol is actually a tuple of local protocols, one for each agent in the system, and that a local protocol consists of state, message, and action protocols. We will define the composition of two message protocols A and B. The composition of message and action protocols is similar, and the composition of local protocols and protocols will immediately follow.

We can assume without loss of generality that the domains $dom(A)$ and $dom(B)$ of A and B (that is, the sets of local states on which the functions

A and B are defined) are disjoint. Let $halt(A)$ and $start(B)$ be the halt states of A and start states of B, respectively. The only real problem in the definition of $A;B$ is how the composition should move from a halt state of A to a start state of B. In the case of interactive protocols, for example, it seems most natural to require that the states of the communication tapes, work tapes, and random tapes encoded in a local state remain the same, and that the only thing that changes is that the state of the Turing machine describing the prover or verifier's protocol changes from a halt state of the first protocol to the start state of the second. This can be described by a function f from $halt(A)$ to $start(B)$. The sequential composition $A;B$ of A and B, given f, is defined by

$$A;B(s,\vec{m}) = \begin{cases} A(s,\vec{m}) & \text{if } s \in dom(A) - halt(A) \\ f(s) & \text{if } s \in halt(A) \\ B(s,\vec{m}) & \text{if } s \in dom(B) \end{cases}$$

(Remember that a state protocol A maps a local state s and a vector $\vec{m}$ of messages received from other protocols to a local state $A(s,\vec{m})$.)

Having defined sequential composition, we now show that the sequential composition of two interactive proofs is an interactive proof. Suppose (P_1, V_1) and (P_2, V_2) are interactive proofs for L_1 and L_2, respectively. Recall that we assume the prover and verifier maintain on their work tapes a complete history of the local states they pass through during the course of a run. Notice that a trivial modification of these proof systems results in proof systems for the languages $\hat{L}_1 = L_1 \times \Sigma^*$ and $\hat{L}_2 = \Sigma^* \times L_2$, respectively, where $\Sigma = \{0,1\}$. Let us abuse notation and denote these new proof systems by (P_1, V_1) and (P_2, V_2) as well. Finally, let $(\hat{P}, \hat{V}) = (P_1; P_2, V_1; V_2)$ be the sequential composition of the two proof systems. We now sketch a proof that $(\hat{P}, \hat{V})$ is an interactive proof system for $\hat{L} = L_1 \times L_2$.

First, we note that it is easy to prove the following:

Claim 5.13:

$$\mathcal{P} \times \hat{V} \models (x \in \hat{L}_1 \wedge \text{`}p \textit{ running } \hat{P}\text{'}) \supset \Diamond \tilde{K}_v^{\psi}(x \in \hat{L}_1)$$

where $\psi \stackrel{def}{=} halted \wedge \text{`}p \textit{ running } \hat{P}\text{'}$.

To see this, notice that since (P_1, V_1) is an interactive proof for $\hat{L}_1$, Proposition 5.3 says

$$\mathcal{P} \times V_1 \models (x \in \hat{L}_1 \wedge \text{`}p \textit{ running } P_1\text{'}) \supset \Diamond \tilde{K}_v^{\psi_1}(x \in \hat{L}_1)$$

where $\psi_1 \stackrel{def}{=} halted \wedge$ 'p *running* P_1'. It is clear that any test M in $\mathcal{P} \times V_1$ for $x \in \hat{L}_1$ that is practically sound and practically complete given ψ_1 can be extended to a test $\hat{M}$ in $\mathcal{P} \times \hat{V}$ that is sound and practically complete given ψ: the test $\hat{M}$ simply searches its work tape for the most recent local state in which the verifier was running V_1, runs M in this state, and accepts iff M accepts.

It is a bit harder to prove that

Claim 5.14:

$$\mathcal{P} \times \hat{V} \models (x \in \hat{L}_2 \wedge \text{'}p \textit{ running } \hat{P}\text{'}) \supset \Diamond \tilde{K}_v^{\psi}(x \in \hat{L}_2),$$

where $\psi \stackrel{def}{=} halted \wedge$ 'p *running* $\hat{P}$'.

To prove this, we observe that since (P_2, V_2) is an interactive proof for $\hat{L}_2$, Proposition 5.3 says

$$\mathcal{P} \times V_2 \models (x \in \hat{L}_2 \wedge \text{'}p \textit{ running } P_2\text{'}) \supset \Diamond \tilde{K}_v^{\psi_2}(x \in \hat{L}_2)$$

where $\psi_2 \stackrel{def}{=} halted \wedge$ 'p *running* P_2'. We want to say that any test M in $\mathcal{P} \times V_2$ for $x \in \hat{L}_2$ that is sound and complete given ψ_2 can be extended to a test $\hat{M}$ in $\mathcal{P} \times \hat{V}$ for $x \in \hat{L}_2$. The test $\hat{M}$ is defined as follows. Since $\hat{V} = V_1; V_2$, it is easy to see that there is a natural mapping h mapping a point c of $\mathcal{P} \times \hat{V}$ in which the verifier is running V_2 to a point d of $\mathcal{P} \times V_2$. This mapping essentially discards that portion of a run of $\mathcal{P} \times \hat{V}$ up to the point V_2 is started, erasing everything on the communication and random tapes that is written before the beginning of V_2, leaving the input and work tapes unchanged. The test $\hat{M}$ rejects at a point if the verifier is still following V_1, and at all other points c runs the test M on the point $h(c)$. The problem is showing that $\hat{M}$ is practically sound and practically complete given ψ.

To do this, we have to relate the probability spaces used in $\mathcal{P} \times \hat{V}$ to evaluate formulas like $pr[\varphi] \geq \alpha$ to the probability spaces used in $\mathcal{P} \times V_2$. It is easy to see that, extending h to sets in the obvious way, h maps $S_{i,c}$ to $S_{i,d}$ (where $d = h(c)$) and measurable sets of $S_{i,c}$ to measurable sets of $S_{i,d}$ with the same measure. Furthermore, the fact $x \in L$ holds at c iff it does at $h(c)$, and the test $\hat{M}$ accepts with the same probability at both c and $h(c)$. Consequently, the fact that $init \supset pr[\varphi] \geq \alpha$ is valid in $\mathcal{P} \times V_2$, where φ is of the form $sound(M, K_v(x \in L))$, implies that $init \supset \Diamond(pr[\varphi] \geq \alpha)$ is valid

in $\mathcal{P} \times \hat{V}$, and hence that $init \supset pr[\varphi] \geq \alpha$ is valid in $\mathcal{P} \times \hat{V}$. Consequently, the fact that M is practically sound in $\mathcal{P} \times V_2$ implies that $\hat{M}$ is practically sound in $\mathcal{P} \times \hat{V}$, and similarly for practical completeness given ψ. This proves Claim 5.14.

Given the two Claims 5.13 and 5.14, we know that the two formulas

$$(x \in \hat{L}_1 \wedge \text{`}p \textit{ running } \hat{P}\text{'}) \supset \Diamond \tilde{K}_v^{\psi}(x \in \hat{L}_1)$$

and

$$(x \in \hat{L}_2 \wedge \text{`}p \textit{ running } \hat{P}\text{'}) \supset \Diamond \tilde{K}_v^{\psi}(x \in \hat{L}_2)$$

are valid in $\mathcal{P} \times \hat{V}$. Notice that $x \in \hat{L}$ implies $x \in \hat{L}_1$ and $x \in \hat{L}_2$, and that, since $K_v^{\psi}(x \in \hat{L}_1)$ and $K_v^{\psi}(x \in \hat{L}_2)$ are stable formulas (once they become true they remain true), $\Diamond K_v^{\psi}(x \in \hat{L}_1) \wedge \Diamond K_v^{\psi}(x \in \hat{L}_2)$ implies $\Diamond K_v^{\psi}(x \in \hat{L})$. It follows that

Corollary 5.15:

$$\mathcal{P} \times \hat{V} \models (x \in \hat{L} \wedge \text{`}p \textit{ running } \hat{P}\text{'}) \supset \Diamond \tilde{K}_v^{\hat{\psi}}(x \in \hat{L}),$$

where $\psi \stackrel{def}{=} halted \wedge \text{`}p \textit{ running } \hat{P}\text{'}$.

Finally, by Proposition 5.4 we have

Proposition 5.16: The interactive protocol $(\hat{P}, \hat{V})$ can be effectively modified to obtain an interactive proof for the language $\hat{L}$.

5.9 Conclusion

The main contribution of this work lies in suggesting notions of knowledge appropriate for interactive proofs, characterizing interactive proofs in terms of these notions, and proving, again in terms of these notions, that the prover in a zero knowledge proof system does not leak any information other than the fact it set out to prove. Roughly speaking, we have shown that a zero knowledge proof system for $x \in L$ satisfies the following property, which we call *knowledge security*: the prover is guaranteed that, with high probability, if the verifier will practically know a fact φ at the end of the proof, it practically knows $x \in L \supset \varphi$ at the start. We have also formalized the notion of

knowing how to generate, and shown that zero knowledge proofs also satisfy an analogous property of *generation security*. (The precise formulations of knowledge and generation security are provided by the statements of Theorems 5.5 and 5.7.) It is currently an open question whether either of these notions of security characterizes zero knowledge (that is, say, whether an interactive proof that satisfies the property of knowledge security is also a zero knowledge proof). We can show, however, that, in the context of finite state protocols, any protocol that satisfies the knowledge security property is *recognition zero knowledge*, as defined in [DS88]. We consider the problem of characterizing zero knowledge in terms of knowledge instead of simply stating necessary conditions for zero knowledge (knowledge and generation security) to be an important problem.

We have sketched in Section 5.8 an example of how practical knowledge can be used to reason about cryptographic protocols like interactive proof systems. A second important problem left unsolved by this chapter is that of developing more sophisticated tools for reasoning about practical knowledge (and, for that matter, knowing how to generate) that will be needed in order to be able to prove more sophisticated results about cryptography in terms of knowledge. In Chapter 3 we were able to use fairly powerful proof rules like the induction rule to reason about information-theoretic definitions of knowledge, a rule that is essentially the translation of theorems from recursion theory into statements about knowledge. In the case of probabilistic knowledge, it is possible to translate many results theorems about measure theory into proof rules for probabilistic knowledge (see [FH88] for a number of examples). But because the definition of practical knowledge depends on Turing machines, powerful proof rules for reasoning about practical knowledge are going to require general results about computation and computational complexity. Some simple proof rules such as "*From* $K_q^{\psi_1}\varphi_1$ *and* $K_q^{\psi_2}\varphi_2$ *infer* $K_q^{\psi_1\wedge\psi_2}(\varphi_1 \wedge \varphi_2)$" are quite easy to prove valid. But we have seen in Section 5.3.2 and the work of [Mos88] that proof rules such as "*From* $K_q^{\psi}\varphi$ *and* $K_q^{\psi}(\varphi \supset \varphi')$ *infer* $K_q^{\psi}\varphi'$" are not necessarily valid. Under what conditions are such rules valid? It is not clear at the moment how different reasoning about such conditions and using the resulting proof rules will be from making such inferences by reasoning directly in terms of the operational, cryptographic definitions in the first place. Moreover, we want to be able to reason about interactive protocols in isolation, and use these results to reason about protocols making use of interactive protocols

as subroutines. This means that we want to be able to prove that certain statements about knowledge are valid in a system corresponding to running an interactive protocol in isolation, and prove that these same statements are true in another system at all points at which the interactive proof is being run as a subroutine. But we do not seem to have at the moment very sophisticated techniques for translating statements about knowledge from one system to another, although the mapping h used in Section 5.8 and the related notions of implementation defined by Halpern and Fagin in [HF85] and elaborated by Mazer in [Maz89] are a good initial step toward this goal.

Nonetheless, we feel that these security results shed some light on the type of security that zero knowledge proofs provide. Our theorems provide support for the definitions of interactive proofs and zero knowledge and our model provides a good semantic setting for such an analysis. Some of the definitions, chiefly that of practical knowledge, are quite subtle. Many straightforward definitions one may try fail by being inappropriate for the cryptographic setting and not providing a useful sense in which zero knowledge proof systems provide security. As Feige, Fiat, and Shamir write in [FFS87], "the notion of 'knowledge' is very fuzzy, and *a priori* it is not clear what proofs of knowledge actually prove." We hope to have established a framework within which such questions can now be answered.

5.A Proofs of results

We end this chapter with an appendix in which we prove most of the results claimed in this chapter. As stated in the text, the proofs of the remaining results either follow immediately from preceding results, or are virtually identical to the proofs of the preceding results.

Proposition 5.1: An interactive protocol (P, V) is an interactive proof system for a language L iff the following conditions are satisfied:

- *Completeness*: For every $k \geq 1$ there exists $\alpha \geq 1$ such that

$$P \times V \models \mathit{init} \supset Pr[x \in L \supset \Diamond \mathit{accept}] \geq 1 - \alpha\,|x|^{-k}.$$

- *Soundness*: For every $k \geq 1$ there exists $\alpha \geq 1$ such that

$$P \times V \models \mathit{init} \supset Pr[\Diamond \mathit{accept} \supset x \in L] \geq 1 - \alpha\,|x|^{-k}.$$

Proof: First, given an interactive proof system (P, V) for a language L, we prove that the two conditions above are satisfied. Fix $k \geq 1$, let $N_k \geq 1$ be the constant guaranteed by the definition of an interactive proof system, and take $\alpha = (N_k)^k$; notice that $1 - \alpha |x|^{-k} \leq 0$ when $|x| < N_k$.

We first prove that the completeness condition is satisfied. It is enough to show that for any initial point c of $P \times V$, the point c satisfies the formula ψ_1 defined by $Pr[x \in L \supset \Diamond accept] \geq 1 - \alpha |x|^{-k}$. Fix one such point c. Notice that fixing c implies fixing an initial global state, and hence fixing values for x, s, and t. If $x \notin L$, then all points with c's global state satisfy the formula $x \in L \supset \Diamond accept$, and hence c satisfies ψ_1. Suppose $x \in L$. If $|x| < N_k$, then by the choice of α we have $1 - \alpha |x|^{-k} < 0$, and c trivially satisfies ψ_1. If $|x| \geq N_k$, then by the completeness condition for interactive proof systems we have that the verifier accepts in $1 - |x|^{-k} \geq 1 - \alpha |x|^{-k}$ of the runs of $(P(s), V(t))(x)$; in other words, $\Diamond accept$ holds at $1 - \alpha |x|^{-k}$ of the points with c's global state, and c satisfies ψ_1.

We now show the soundness condition is satisfied: Again, it is enough to show that for any initial point c of $\mathcal{P} \times V$, the point c satisfies the formula ψ_2 defined by $Pr[\Diamond accept \supset x \in L] \geq 1 - \alpha |x|^{-k}$. Fix one such point c. Again, notice that fixing c implies fixing an initial global state, and hence fixing values for P^*, x, s, and t. If $x \in L$, then all points with c's global state satisfy the formula $\Diamond accept \supset x \in L$, and hence c satisfies ψ_2. Suppose $x \notin L$. If $|x| < N_k$, then by the choice of α we have $1 - \alpha |x|^{-k} < 0$, and c trivially satisfies ψ_2. If $|x| \geq N_k$, by the soundness condition for interactive proof systems it follows that the verifier accepts in at most $|x|^{-k}$ of the runs of P^* and V on input x with work tapes s and t. This means that at least $1 - |x|^{-k} \geq 1 - \alpha |x|^{-k}$ of the points with c's global state fail to satisfy $\Diamond accept$, and hence must satisfy $\Diamond accept \supset x \in L$. It follows that c satisfies ψ_2.

Conversely, given (P, V) satisfying the two conditions above, we prove (P, V) is an interactive proof system for L. Fix $k \geq 1$, let $\alpha \geq 1$ be the constant guaranteed by the two conditions above for $2k$, and take $N_k \geq 1$ to be large enough that $\alpha < (N_k)^k$; notice that $\alpha < |x|^k$ when $|x| \geq N_k$.

We first show the completeness condition for an interactive proof system is satisfied. Consider any x, s, and t satisfying $x \in L$ and $|x| \geq N_k$. The completeness condition above guarantees, in particular, that the verifier accepts in at least $1 - \alpha |x|^{-2k}$ of the runs of P and V on input x with work tapes s and t. Since the choice of α guarantees $1 - \alpha |x|^{-2k} \geq 1 - |x|^{-k}$, we

have $Pr[(P(s), V(t))(x) \text{ accepts}] \geq 1 - |x|^{-k}$.

We now prove the soundness condition for an interactive proof system is satisfied. Consider any P^*, x, s, and t satisfying $x \notin L$ and $|x| \geq N_k$. Since $x \notin L$, the soundness condition above guarantees that the verifier fails to accept in at least $1 - \alpha\,|x|^{-2k}$ of the runs with P^* and V on input x with work tapes s and t, which means the verifier accepts in at most $\alpha\,|x|^{-2k} \leq |x|^{-k}$ runs, so $Pr[(P^*(s), V(t))(x) \text{ accepts}] \leq |x|^{-k}$. □

Proposition 5.3: If (P, V) is an interactive proof system for L, then

$$\mathcal{P} \times V \models (x \in L \wedge \text{`}p \textit{ running } P\text{'}) \supset \Diamond \tilde{K}_v^{\psi}(x \in L),$$

where $\psi \stackrel{def}{=} \textit{halted} \wedge \text{`}p \textit{ running } P\text{'}$.

Proof: Let M be the test that accepts at a point if the verifier has accepted at that point, and rejects otherwise. Suppose we can show that M is practically sound for $K_v(x \in L)$, and practically complete for $K_v(x \in L)$ given ψ. Then we can complete the proof of this proposition as follows. Consider any point (r, k) of $\mathcal{P} \times V$ satisfying $x \in L \wedge \text{`}p \textit{ running } P\text{'}$, and consider any final point (r, k') of r with $k' \geq k$. Notice that $(r, k') \models \psi$ and $(r, k) \models K_v(x \in L)$. Since M is a test for $K_v(x \in L)$ that is sound and is complete given ψ, we have $(r, k') \models \tilde{K}_v^{\psi}(x \in L)$, and hence $(r, k) \models \Diamond \tilde{K}_v^{\psi}(x \in L)$. It follows that

$$\mathcal{P} \times V \models (x \in L \wedge \text{`}p \textit{ running } P\text{'}) \supset \Diamond \tilde{K}_v^{\psi}(x \in L),$$

as desired. Thus, all we need to prove is that M is practically sound for $K_v(x \in L)$, and practically complete for $K_v(x \in L)$ given ψ. Since $K_v(x \in L)$ is equivalent to $x \in L$, it is enough to prove that M is practically sound for $x \in L$, and practically complete for $x \in L$ given ψ.

To see that M is practically sound for $x \in L$, fix $k \geq 1$ and take $\alpha \geq 1$ to be the constant guaranteed by Proposition 5.1 to satisfy

$$\mathcal{P} \times V \models \textit{init} \supset Pr[\Diamond \textit{accept} \supset x \in L] \geq 1 - \alpha\,|x|^{-k}.$$

Notice that the formula $\Diamond \textit{accept} \supset x \in L$ implies $x \notin L \supset \neg \textit{accept}$, which in turn implies $\textit{sound}(M, x \in L)$. Since $\Diamond \textit{accept} \supset x \in L$ is a fact about the run, $\Diamond \textit{accept} \supset x \in L$ implies $\Box(\Diamond \textit{accept} \supset x \in L)$, which in turn implies $\Box \textit{sound}(M, x \in L)$. It follows that

$$\mathcal{P} \times V \models \textit{init} \supset Pr[\Box \textit{sound}(M, x \in L)] \geq 1 - \alpha\,|x|^{-k},$$

and hence M is sound for $x \in L$.

To see that M is practically complete for $x \in L$ given ψ, fix $k \geq 1$ and take $\alpha \geq 1$ be the constant guaranteed by Proposition 5.1 to satisfy

$$P \times V \models \mathit{init} \supset Pr[x \in L \supset \Diamond \mathit{accept}] \geq 1 - \alpha\,|x|^{-k}.$$

Notice that the formula $x \in L \supset \Diamond \mathit{accept}$ implies $\psi \supset (x \in L \supset (\psi \wedge \Diamond \mathit{accept}))$. Since the formula $\psi \wedge \Diamond \mathit{accept}$ is equivalent to *accept* (the verifier has already accepted or rejected at points satisfying ψ, namely final points), and since $x \in L \supset \mathit{accept}$ implies $\mathit{complete}(M, x \in L)$, we have $\psi \supset \mathit{complete}(M, x \in L)$. Finally, since $x \in L \supset \Diamond \mathit{accept}$ is a fact about the run, $x \in L \supset \Diamond \mathit{accept}$ implies $\Box[x \in L \supset \Diamond \mathit{accept}]$, which implies $\Box[\psi \supset \mathit{complete}(M, x \in L)]$. It follows that

$$P \times V \models \mathit{init} \supset Pr[\Box[\psi \supset \mathit{complete}(M, x \in L)]] \geq 1 - \alpha\,|x|^{-k}.$$

But we want to prove that this formula is valid in the system $\mathcal{P} \times V$, and not $P \times V$. Since a point of $\mathcal{P} \times V$ satisfying ψ is a point of $P \times V$ (recall that $\psi \supset$ '*p running P*'), we have

$$\mathcal{P} \times V \models \mathit{init} \supset Pr[\Box[\psi \supset \mathit{complete}(M, x \in L)]] \geq 1 - \alpha\,|x|^{-k},$$

as desired, and hence M is complete for $x \in L$ given ψ. $\Box$

Proposition 5.4: If

$$\mathcal{P} \times V^* \models (x \in L \wedge \text{'}p \textit{ running } P\text{'}) \supset \Diamond \tilde{K}_v^{\psi}(x \in L),$$

where $\psi \stackrel{\mathit{def}}{=} \mathit{halted} \wedge$ '*p running P*', then we can effectively modify V^* to obtain V such that (P, V) is an interactive proof system for L.

Proof: Let M be a test for $K_v(x \in L)$, and hence for $x \in L$, that is practically sound, and practically complete given ψ. Such a test M is guaranteed to exist by the definition of practical knowledge given ψ. We assume without loss of generality that M accepts with probabilities $2^{-|x|}$ and $1 - 2^{-|x|}$ instead of $1/3$ and $2/3$.[9] Let V be the protocol in which the verifier (i) runs the

[9] We can always transform a test M accepting with probabilities $1/3$ and $2/3$ into a test M' accepting with probabilities $2^{-|x|}$ and $1 - 2^{-|x|}$ by using the standard trick of running the test M many times to estimate the probability with which M accepts or rejects.

protocol V^*, (ii) runs the test M once V^* halts, and (iii) accepts iff M accepts. We now show that (P, V) satisfies the soundness and completeness of Proposition 5.1, and hence must be an interactive proof system for L. Given a run r of $\mathcal{P} \times V$ and a run r^* of $\mathcal{P} \times V^*$, we say that r and r^* are *corresponding runs* if the two runs have the same initial state, and the sequences of coins flipped in the two runs are the same. We say that (r, k) and (r^*, k) are *corresponding points.*

We first prove that (P, V) satisfies the soundness condition

$$\mathcal{P} \times V \models \mathit{init} \supset Pr[\Diamond \mathit{accept} \supset x \in L] \geq 1 - \alpha\,|x|^{-k}.$$

of Proposition 5.1. Since M is practically sound for $x \in L$ in $\mathcal{P} \times V^*$, we have

$$\mathcal{P} \times V^* \models \mathit{init} \supset Pr(\Box \mathit{sound}(M, x \in L)) \geq 1 - \alpha\,|x|^{-k}.$$

Recall that $\mathit{sound}(M, x \in L)$ holds at a point if at that point $x \notin L$ implies $pr[M\ \mathit{rejects}] \geq 1 - 2^{-|x|}$. Remember that the probability here is being taken over M's coin flips (and not over runs), and that this condition is a fact about the global state (even a fact about the verifier's local state, the input to the test M). If we take this condition as a primitive proposition in our language, then $\mathit{sound}(M, x \in L)$ is equivalent to the formula $x \notin L \supset pr[M\ \mathit{rejects}] \geq 1 - 2^{-|x|}$. It follows that

$$\Box \mathit{sound}(M, x \in L)$$

implies

$$\Box(x \notin L \supset pr[M\ \mathit{rejects}] \geq 1 - 2^{-|x|}).$$

We claim that, given corresponding runs r and r^* of $\mathcal{P} \times V$ and $\mathcal{P} \times V^*$, if the initial point $(r^*, 0)$ satisfies $\Box \mathit{sound}(M, x \in L)$ and hence satisfies

$$\Box(x \notin L \supset pr[M\ \mathit{rejects}] \geq 1 - 2^{-|x|}),$$

then the initial point $(r, 0)$ satisfies

$$x \notin L \supset \Diamond(Pr[\Diamond \mathit{reject}] \geq 1 - 2^{-|x|}).$$

To see this, let ℓ be the time at which the verifier has finished the protocol V^* in r and r^* and starts the test M in r. If $(r^*, 0) \models x \notin L$, then $(r^*, \ell) \models pr[M\ \mathit{rejects}] \geq 1 - 2^{-|x|}$. Consequently, if $(r, 0) \models x \notin L$, then $(r, \ell) \models$

$Pr[\Diamond reject] \geq 1 - 2^{-|x|}$. (Remember that the probability is being taken over M's coin flips at (r^*, ℓ) and over runs at (r, ℓ).) It follows that $(r, 0)$ satisfies the formula $x \notin L \supset \Diamond(Pr[\Diamond reject] \geq 1 - 2^{-|x|})$, as desired.

Now let $(r, 0)$ be any initial point of $\mathcal{P} \times V$, and let $(r^*, 0)$ be the corresponding initial point of $\mathcal{P} \times V^*$. Since the soundness of M guarantees that the initial point $(r^*, 0)$ must satisfy the formula $Pr[\Box sound(M, x \in L))] \geq 1 - \alpha |x|^{-k}$, the preceding argument shows that the initial point $(r, 0)$ must satisfy the formula

$$Pr[x \notin L \supset \Diamond(Pr[\Diamond reject] \geq 1 - 2^{-|x|})] \geq 1 - \alpha |x|^{-k}.$$

It follows that $(r, 0)$ satisfies

$$Pr[x \notin L \supset \Diamond reject] \geq (1 - 2^{-|x|})(1 - \alpha |x|^{-k}),$$

which implies

$$Pr[\Diamond accept \supset x \in L] \geq (1 - 2^{-|x|})(1 - \alpha |x|^{-k}).$$

Since

$$\begin{aligned}(1 - 2^{-|x|})(1 - \alpha |x|^{-k}) &\geq 1 - \alpha |x|^{-k} - 2^{-|x|} \\ &= 1 - \alpha |x|^{-k}(1 + \frac{2^{-|x|}}{\alpha |x|^{-k}}) \\ &\geq 1 - \alpha\beta |x|^{-k}\end{aligned}$$

for some $\beta \geq 1$, it follows that $(r, 0)$ satisfies

$$Pr[\Diamond accept \supset x \in L] \geq 1 - \gamma |x|^{-k}$$

for some $\gamma \geq 1$. Thus, (P, V) satisfies the soundness condition.

We now prove that (P, V) satisfies the completeness condition

$$P \times V \models init \supset Pr[x \in L \supset \Diamond accept] \geq 1 - \alpha |x|^{-k}$$

of Proposition 5.1. Since M is complete for $x \in L$ given ψ in $\mathcal{P} \times V^*$, we have

$$\mathcal{P} \times V^* \models init \supset Pr[\Box(\psi \supset complete(M, x \in L))] \geq 1 - \alpha |x|^{-k}.$$

As above, taking $pr[M\ accepts] \geq 1 - 2^{-|x|}$ as a primitive proposition in our language, the condition $complete(M, x \in L)$ is equivalent to the formula $x \in L \supset pr[M\ accepts] \geq 1 - 2^{-|x|}$. It follows that

$$\Box(\psi \supset complete(M, x \in L))$$

implies

$$\Box(\psi \supset (x \in L \supset pr[M\ accepts] \geq 1 - 2^{-|x|}))$$

which implies

$$x \in L \supset \Box(\psi \supset pr[M\ accepts] \geq 1 - 2^{-|x|})$$

since $x \in L$ is a fact about the run.

We claim that, given corresponding runs r and r^* of $P \times V$ and $P \times V^*$, if the initial point $(r^*, 0)$ satisfies $\Box(\psi \supset complete(M, x \in L))$ and hence satisfies

$$x \in L \supset \Box(\psi \supset pr[M\ accepts] \geq 1 - 2^{-|x|}),$$

then the initial point $(r, 0)$ satisfies

$$x \in L \supset \Diamond(Pr[\Diamond accept] \geq 1 - 2^{-|x|}).$$

To see this, let ℓ be the time at which the verifier has finished the protocol V^* in r and r^* and starts the test M in r. If $(r^*, 0) \models x \in L$, then $(r^*, \ell) \models pr[M\ accepts] \geq 1 - 2^{-|x|}$ since $(r^*, \ell) \models \psi$. Consequently, if $(r, 0) \models x \in L$, then $(r, \ell) \models Pr[\Diamond accept] \geq 1 - 2^{-|x|}$, and hence $(r, 0)$ satisfies the formula $x \in L \supset \Diamond(Pr[\Diamond accept] \geq 1 - 2^{-|x|})$.

Now let $(r, 0)$ be any initial point of $P \times V$, and let $(r^*, 0)$ be the corresponding initial point of $P \times V^*$. Since the completeness of M given ψ guarantees that the initial point $(r^*, 0)$ must satisfy the formula

$$Pr[\Box(\psi \supset complete(M, x \in L))] \geq 1 - \alpha\,|x|^{-k},$$

the preceding argument shows that the initial point $(r, 0)$ must satisfy

$$Pr[x \in L \supset \Diamond(Pr[\Diamond accept] \geq 1 - 2^{-|x|})] \geq 1 - \alpha\,|x|^{-k}.$$

It follows that $(r, 0)$ satisfies

$$Pr[x \in L \supset \Diamond accept] \geq (1 - 2^{-|x|})(1 - \alpha\,|x|^{-k}),$$

and hence

$$Pr[x \in L \supset \Diamond accept] \geq 1 - \gamma^{-|x|^{-k}}$$

for some $\gamma \geq 1$ as above. Thus, (P, V) satisfies the completeness condition. □

Theorem 5.5: Let (P, V) be a zero knowledge proof system for L, let V^* be an arbitrary verifier, and let φ be a fact about the initial state. For every fact ψ and constant $k \geq 1$ there is a fact ψ' and a constant $\alpha \geq 1$ such that

$$P \times V^* \models (x \in L \wedge init) \supset K_p^{1-\alpha|x|^{-k}}[\Diamond \tilde{K}_v^{\psi}\varphi \supset \tilde{K}_v^{\psi'}(x \in L \supset \varphi)].$$

Proof: Given a fact ψ and a constant k, we construct a fact ψ' and constant α satisfying the formula above.

Notice that we can assume $\tilde{K}_v^{\psi}\varphi$ holds at some point of $P \times V^*$ (the theorem is trivially true if it does not), and hence the existence of a test M for $K_v\varphi$ that is practically sound and is practically complete given ψ. Without loss of generality we can assume two things about this test. First, we can assume that M accepts with probabilities $2^{-|x|}$ or $1 - 2^{-|x|}$ instead of 1/3 or 2/3. Second, since we assume that the verifier's local state encodes the verifier's local history, and since φ is a fact about the initial state, if $K_v\varphi$ holds at any point of a proof then it holds at the end of the proof as well. Consequently, since the verifier's local state does encode the verifier's history, we can assume that M accepts with probability 2/3 at the end of a proof if it does so at any point in the middle of the proof. Neither assumption affects the fact that M is practically sound for $K_v\varphi$, and practically complete for $K_v\varphi$ given ψ. Given the constant k fixed above, let α_{3k} be the constant guaranteed for $3k$ by the definition of the practical soundness and completeness of M.

We can also assume the existence of a Turing machine $M_{V^*}(t, x)$ that approximates the distribution of local histories generated by $(P(s), \underline{V}^*(t))(x)$. In particular, the following modification M_h of the test M is able to distinguish these distributions with only negligible probability. Notice that the input to M is the verifier's local state. We can modify M to obtain a test M_h that accepts as input the verifier's local history and runs the test M at the final local state in the local history, accepting iff the test M accepts. Since the length of the interactive proof is bounded by some polynomial in $|x|$, we can guarantee that M_h still runs in time polynomial in $|x|$ on arbitrary inputs by having it reject outright when presented with a history that is too long.

Consider now the test T' defined as follows:

```
T'(t,x):  accepted := false
          repeat 6|x|^k times
              run M_{V*}(t,x) to generate a local history H
              if M_h accepts H then
                  accepted := true
          end repeat;
          if accepted then accept else reject.
```

In a few moments we will prove that T' is a test for $K_v(x \in L \supset \varphi)$ that, for some constant X, is sound at all points with $|x| \geq X$, and is complete at all points with $|x| \geq X$ that satisfy

$$\psi'_x \stackrel{def}{=} init \quad \wedge \quad x \in L \quad \wedge \quad Pr[\Diamond \tilde{K}_v^{\psi} \varphi] \geq |x|^{-k}.$$

In fact, we will show that T' accepts with probability 2/3 at *all* points with $|x| \geq X$ that satisfy ψ'_x. Taking ψ' to be the fact holding at points satisfying $|x| \geq X$ and ψ'_x, and taking T to be the test obtained by modifying T' to reject outright if $|x| < X$, it will follow that T is a test for $K_v(x \in L \supset \varphi)$ that is sound and is complete given ψ'. In fact, T will accept with probability 2/3 at all points satisfying ψ'.

Given such a test T, the rest of the proof is completed as follows. Take $\alpha = X^k$ so that $1 - \alpha |x|^{-k} \leq 0$ when $|x| < X$. Consider an initial point c satisfying $x \in L$. If c satisfies $Pr[\Diamond \tilde{K}_v^{\psi} \varphi] < |x|^{-k}$, then c trivially satisfies $Pr[\Diamond \tilde{K}_v^{\psi} \varphi \supset \tilde{K}_v^{\psi'}(x \in L \supset \varphi)] \geq 1 - |x|^{-k}$. If c satisfies $|x| < X$, then $1 - \alpha |x|^{-k} < 0$, and c trivially satisfies $Pr[\Diamond \tilde{K}_v^{\psi} \varphi \supset \tilde{K}_v^{\psi'}(x \in L \supset \varphi)] \geq 1 - \alpha |x|^{-k}$. So suppose c satisfies $Pr[\Diamond \tilde{K}_v^{\psi} \varphi] \geq |x|^{-k}$ and $|x| \geq X$. Notice that c satisfies ψ', and hence that T accepts with probability 2/3 at c. Since T is sound for $K_v(x \in L \supset \varphi)$, it follows that c satisfies $K_v(x \in L \supset \varphi)$, and hence that c satisfies $Pr[\Diamond \tilde{K}_v^{\psi} \varphi \supset \tilde{K}_v^{\psi'}(x \in L \supset \varphi)] = 1$. Consequently, all initial points c with $x \in L$ satisfy $Pr[\Diamond \tilde{K}_v^{\psi} \varphi \supset \tilde{K}_v^{\psi'}(x \in L \supset \varphi)] \geq 1 - \alpha |x|^{-k}$, and hence satisfy $K_p^{1-\alpha|x|^{-k}}[\Diamond \tilde{K}_v^{\psi} \varphi \supset \tilde{K}_v^{\psi'}(x \in L \supset \varphi)]$ as desired.

It remains only to prove that, for some constant X, the test T' is sound at points with $|x| \geq X$ and is complete at points with $|x| \geq X$ satisfying ψ'_x.

We first prove that T' is sound at all points with sufficiently large x: given a point c of $P \times V^*$ satisfying $\neg K_v(x \in L \supset \varphi)$ with sufficiently large x, we prove that T' rejects with probability 2/3 at c.

Since c satisfies $\neg K_v(x \in L \supset \varphi)$, some point c' of $\mathcal{P} \times V$ with $c \sim_v c'$ satisfies $\neg(x \in L \supset \varphi)$. Since T' takes as input only x and t found in the v's

local state, which is the same at both c and c', the test T' must reject with the same probability at both points. Without loss of generality, therefore, we can assume c satisfies $\neg(x \in L \supset \varphi)$, or equivalently that c satisfies $x \in L$ but not φ.

T' rejects at c iff, on each iteration, M_h rejects a history generated by M_{V^*}. What is the probability that M_h rejects a history generated by M_{V^*}? Suppose the point c fixed above is the initial point of a run of $(P(s), V^*(t))(x)$. Since (P, V) is a zero-knowledge proof system, we know that, for sufficiently large x, the probability M_h rejects a history generated by $M_{V^*}(t, x)$ is within $|x|^{-2k}$ of the probability M_h rejects a history generated by $(P(s), \underline{V}^*(t))(x)$. But this latter probability is just the probability the original test M rejects at the end of a run of $(P(s), V^*(t))(x)$. Since c satisfies $\neg\varphi$, and since φ is a fact about the initial state, we know that $\neg\varphi$, and hence $\neg K_v\varphi$, holds at all points of every run of $(P(s), V^*(t))(x)$. Since M is practically sound for $K_v\varphi$, we know that, for sufficiently large x, the test M rejects with probability at least $1 - 2^{-|x|}$ at the end of at least $1 - \alpha_{3k}|x|^{-3k} \geq 1 - |x|^{-2k}$ of the runs of $(P(s), V^*(t))(x)$. Consequently, the probability M_h rejects a history generated by $M_{V^*}(t, x)$, and hence the probability a given iteration of T' rejects at c, is at least

$$\begin{aligned}
(1 - 2^{-|x|})(1 - |x|^{-2k}) - |x|^{-2k} &\geq 1 - 2|x|^{-2k} - 2^{-|x|} \\
&\geq 1 - |x|^{-2k}\left(2 + \frac{2^{-|x|}}{|x|^{-2k}}\right) \\
&\geq 1 - 3|x|^{-2k}
\end{aligned}$$

for sufficiently large x; and hence the probability T' rejects at c (that is, that all $6|x|^k$ iterations of T' reject) is at least $(1 - 3|x|^{-2k})^{6|x|^k}$, which goes to 1 as $|x|$ goes to infinity. It follows that T' rejects with probability 2/3 at c for sufficiently large x.

We now prove that T' is complete at all points satisfying ψ'_x with sufficiently large x: given a point c of $P \times V^*$ satisfying ψ'_x with sufficiently large x, we prove that T' accepts with probability 2/3 at c.

First consider the probability a given iteration of T' accepts at c. Suppose the given point c is an initial point of a run of $(P(s), V^*(t))(x)$. Since (P, V) is a zero-knowledge proof system, we know that, for sufficiently large x, the probability M_h accepts a history generated by $M_{V^*}(t, x)$ is within $|x|^{-2k}$ of the probability M_h accepts a history generated by $(P(s), \underline{V}^*(t))(x)$, which is

precisely the probability the original test M accepts at the end of a run of $(P(s), V^*(t))(x)$. Since c satisfies ψ'_x, c satisfies $Pr[\Diamond \tilde{K}_v^{\psi}\varphi] \geq |x|^{-k}$. This means that at least $|x|^{-k}$ of the runs of $(P(s), V^*(t))(x)$ pass through a point satisfying ψ and $K_v\varphi$, and that M accepts with probability at least 2/3 at such points in at least $1 - \alpha_{3k}|x|^{-3k} \geq 1 - |x|^{-k}$ of these runs. Since we assume M accepts with probability 2/3 at the end of a run if it does so in the middle of a run, the same is true at the end of these runs. This means one iteration of T' accepts with probability at least

$$\begin{aligned} \frac{2}{3}|x|^{-k}(1-|x|^{-k}) - |x|^{-2k} &\geq |x|^{-k}\left(\frac{2}{3} - \frac{2}{3}|x|^{-k} - |x|^{-k}\right) \\ &\geq \frac{1}{3}|x|^{-k}. \end{aligned}$$

It follows that a given iteration of T' *rejects* with probability at most $1 - |x|^{-k}/3$, that all iterations of T' reject (in which case T' itself rejects) with probability at most $(1 - |x|^{-k}/3)^{6|x|^k}$, and hence that T' *accepts* with probability at least

$$1 - \left(1 - \frac{1/3}{|x|^k}\right)^{6|x|^k} \approx 1 - (e^{-1/3})^6 \geq \frac{2}{3}$$

for sufficiently large x. (Here we are using the fact that $(1 + c/n)^n$ tends to e^c as n tends to infinity.) It follows that T' accepts with probability 2/3 at c satisfying ψ'_x with sufficiently large x. □

Lemma 5.8: There is a weak interactive proof system for L iff L is in BPP.

Proof: Suppose (P, V) is a weak interactive proof for L. Consider the Turing machine M that on input x simulates $(P, V)(x)$ with empty work tapes. Notice that since both P and V run in polynomial time, so does the Turing machine M. By the definition of a (weak) interactive proof system, if $x \in L$ and x is sufficiently large, then $(P, V)(x)$ and hence $M(x)$ accepts with probability 2/3; and if $x \notin L$ and x is sufficiently large, then $(P, V)(x)$ and hence $M(x)$ rejects with probability 2/3. Since we can hardwire into M whether M should accept or reject x for the finite number of insufficiently large x's, we can assume M is a BPP Turing machine, and hence that L is in BPP.

Conversely, suppose L is in BPP. Let M be a BPP Turing machine for L, and let (P, V) be the interactive protocol defined as follows: on input x, the prover's protocol P does nothing, and the verifier's protocol V runs $M(x)$ and accepts iff $M(x)$ accepts. Since the verifier ignores both the prover and the work tapes, it is clear that for *any* P^*, s, and t, if $x \in L$, then $(P^*(s), V(t))(x)$ accepts with probability 2/3; and if $x \notin L$, then $(P^*(s), V(t))(x)$ rejects with probability 2/3. It follows that (P, V) is a weak interactive proof system for L. □

Lemma 5.9: A weak interactive protocol (P, V) is a weak interactive proof system for a fact φ about the prover's work tape and the common input iff

1. for all sufficiently large x and for all s, we have $\varphi(x, s)$ iff $x \in dom(\varphi)$; and
2. $dom(\varphi)$ is in BPP.

Proof: Suppose (P, V) is a weak interactive proof system for a fact φ about the prover's work tape and the common input. Fix k and let N_k be the constant given by the soundness and complete conditions for a weak interactive proof system.

To prove part 1, suppose for some x with $|x| \geq N_k$ we have $\varphi(x, s)$ and $\neg\varphi(x, s')$, and consider the prover P_s that ignores its work tape and simulates the protocol P on work tape s. Since $\varphi(x, s)$, we know the verifier must accept in $(P(s), V(t))(x)$ with probability at least $1 - |x|^{-k}$. Since $\neg\varphi(x, s')$, we know the verifier must accept in $(P_s(s'), V(t))(x)$ with probability at most $|x|^{-k}$. Notice, however, that the prover P_s on work tape s' simulates the prover P on work tape s, and hence the two distributions $(P(s), V(t))(x)$ and $(P_s(s'), V(t))(x)$ are identical. Consequently, the verifier must accept with the same probability in both $(P(s), V(t))(x)$ and $(P_s(s'), V(t))(x)$, a contradiction. It follows that, for all sufficiently large x with $|x| \geq N_k$, we have $x \in dom(\varphi)$ iff $\varphi(x, s')$ for some s' iff $\varphi(x, s)$ for all s.

To prove part 2, let M be the Turing machine that on input x simulates $(P, V)(x)$ with empty work tapes. Since P and V run in polynomial time, so does M. By part 1 and the definition of a weak interactive proof system (P, V), if $|x| \geq N_k$ and $x \in dom(\varphi)$, then $\varphi(x, \epsilon)$ is satisfied (where ϵ is the empty string), so $(P, V)(x)$ and hence $M(x)$ accepts with probability 2/3; and if $|x| \geq N_k$ and $x \notin dom(\varphi)$, then $\varphi(x, \epsilon)$ is not satisfied, so $(P, V)(x)$

and hence $M(x)$ rejects with probability 2/3. Since we can hardwire into M whether M should accept or reject x for the finite number of insufficiently large x's, we can assume M is a BPP Turing machine, and hence that $dom(\varphi)$ is in BPP.

Conversely, suppose parts 1 and 2 are satisfied. Since $dom(\varphi)$ is in BPP, we know by Lemma 5.8 that there is a weak interactive proof system (P, V) for $dom(\varphi)$. By part 1, for every k and sufficiently large x, if $\varphi(x, s)$ then $x \in dom(\varphi)$ and $(P(s), V(t))(x)$ accepts with probability at least $1 - |x|^{-k}$; and if $\neg\varphi(x, s)$ then $x \notin dom(\varphi)$ $(P^*(s), V(t))(x)$ accepts with probability at most $|x|^{-k}$ for all provers P^*. It follows that (P, V) is a weak interactive proof system for R as well. □

Proposition 5.10: (P, V) is an interactive proof satisfying the correctness condition that the prover can generate a y such that $R(x, y)$ iff (P, V) is a weak interactive proof system for φ_R.

Proof: Suppose (P, V) is an interactive proof satisfying the correctness condition that the prover can generate a y such that $R(x, y)$. We prove that (P, V) is a weak interactive proof for φ_R. For completeness, if $\varphi_R(P, x, s)$ holds then $R(x, s)$ holds, and $(P(s), V(t))(x)$ accepts with high probability by the completeness condition for an interactive proof of [TW87], so (P, V) satisfies the completeness condition for a weak interactive proof of φ_R. For soundness, suppose $\neg\varphi_R(P^*, x, s)$ holds. Since the definition of an interactive proof of [TW87] guarantees that the soundness condition holds for all prover protocols P^*, it is impossible for the fact $\varphi_R(P^*, x, s)$ to be false when $P^* \neq P$. The only way for $\varphi_R(P^*, x, s)$ to be false is when $P^* = P$, in which case the only way for $\varphi_R(P^*, x, s)$ to be false is if $R(x, s)$ is false. In this case, the correctness condition guarantees $(P(s), V(t))(x)$ accepts with low probability, and hence (P, V) satisfies the soundness condition for a weak interactive proof of φ_R.

Conversely, suppose (P, V) is a weak interactive proof for φ_R. We prove that (P, V) is an interactive proof satisfying the correctness condition that the prover can generate a y such that $R(x, y)$. The correctness condition is clearly satisfied, since $\neg R(x, s)$ implies $\neg\varphi_R(P, x, s)$, in which case the soundness condition for a weak interactive proof guarantees $(P(s), V(t))(x)$ accepts with low probability. The completeness condition is also clearly satisfied, since $R(x, s)$ implies $\varphi_R(P, x, s)$, in which case the completeness condition

for a weak interactive proof guarantees $(P(s), V(t))(x)$ accepts with high probability. The definition of φ_R shows the soundness condition is satisfied for prover protocols $P^* \neq P$, so consider the protocol P. Since the completeness condition guarantees that $(P(s), V(t))(x)$ accepts with low probability when $\iota(x, s)$ holds, the trivial generator M_P that simply returns s shows that the soundness condition is satisfied for the prover protocol P as well. Thus, (P, V) is an interactive proof the prover can generate a y such that $R(x, y)$. □

Chapter 6

Conclusion

Since the work of Halpern and Moses [HM84], a number of papers have analyzed problems in distributed computation in terms of knowledge. Our goal has been to apply knowledge to new problems, and to expand the domain of problems to which knowledge can be applied.

The work in Chapter 3 shows how powerful reasoning about knowledge can be. Using the close relationship between common knowledge and simultaneity, we have obtained general, unifying results about computation in unreliable systems. We have identified a general class of problems, including the well-known consensus and distributed firing squad problems, and shown how to transform the specification of such problems into protocols that are optimal in a very strong sense. The state of common knowledge has played a central role in the derivation of these protocols. In the process of implementing tests for common knowledge we have exposed a number of subtle differences between variants of the well-known omissions failure model. This work has shown how knowledge can be used in both protocol design and in the derivation of nontrivial lower bounds on computational complexity. It is not at all clear how the observations leading to these results would have been obtained had we not been thinking about these problems in terms of knowledge.

While this work shows that reasoning about knowledge can be beneficial, we have observed that in some contexts the standard definition of knowledge does not appear to be the most appropriate definition. In the second half of this thesis, we have studied definitions of knowledge for use in two of these contexts.

In the context of probabilistic protocols, the standard definition of knowledge does not enable us to capture a notion of confidence that can be useful when reasoning about such protocols. In Chapter 4, using the framework developed by Fagin and Halpern [FH88], we have examined various definitions of probabilistic knowledge that let us capture several different notions of confidence. We have observed that there is no one notion of confidence that is most appropriate in all contexts. The best way to think about the various definitions is in terms of betting games and betting against different types of adversaries. We have shown, for every given adversary, how to construct the definition of probabilistic knowledge that is provably the best definition in the context of that particular adversary. We have shown how these definitions can be used to analyze a probabilistic variant of the coordinated attack problem.

Cryptography is another context in which the standard definition of knowledge does not capture all relevant aspects of the problems at hand. This is due primarily to the fact that the standard definition does not allow us to express the fact that the bounds on an agent's computational powers affect what that agent can know. In Chapter 5 we have shown how the context of cryptography motivates the definition of *practical knowledge*, a definition of knowledge incorporating both probability and limitations on agents' computational powers. We have show how the definition of practical knowledge can be used to characterize interactive proof systems, and to capture the intuition that a verifier learns essentially nothing as a result of a zero knowledge proof other than the fact the prover initially sets out to prove. Finally, we have sketched how it is possible to reason about such proof systems directly in terms of knowledge, rather than in terms of the operational cryptographic definitions.

While we feel that our work represents significant progress in the attempt to extend the standard definitions of knowledge into other contexts, a number of problems remain. In particular, while we have shown that our definition of practical knowledge can be useful in contexts where agents' computational limitations are of interest, it is by no means clear that it is the most appropriate definition. In fact, it is not even clear what criteria one should use when judging the suitability of a definition in this context. Further progress in this area is of great importance.

While we have noted at the end of each chapter a number of open problems that remain to be resolved, we note that there are two general areas in

which knowledge could possibly play a larger role than it has so far. First, notice that the majority of the results in this thesis have been in the context of synchronous systems. This is generally true in the literature as a whole. The role of knowledge in the context of asynchronous systems has been primarily as a tool for proving lower bounds, but not so much as a tool for the design of new protocols. This is somewhat surprising, since one of the commonly mentioned motivations for formulating definitions of knowledge in the first place is to capture informal statements such as "since p has received message m from q, p knows the task started at q has terminated." Such statements often arise in the context of communications protocols, for example. These protocols are often quite complex, and it would be interesting to know whether a knowledge-based analysis could make such protocols easier to understand, and easier to construct.

Finally, we note that it is becoming increasingly important to be able to reason explicitly about time when designing protocols. For example, timeouts play an important role in the protocols designed for asynchronous systems. Designers often explain these protocols as if the processors themselves must explicitly reason about how their knowledge of the system changes as a result of whether a given timeout occurs or not. It would be interesting to understand how to reason about timeouts (and time in general) directly in terms of formal notions of knowledge. Much remains to be done.

Bibliography

[AUWY82] A. V. Aho, J. D. Ullman, A. D. Wyner, and M. Yannakakis. Bounds on the size and transmission rate of communication protocols. *Computers and Mathematics with Applications*, 8(3):205–214, 1982. This is a later version of [AUY79].

[AUY79] A. V. Aho, J. D. Ullman, and M. Yannakakis. Modeling communication protocols by automata. In *Proceedings of the 20th IEEE Symposium on Foundations of Computer Science*, pages 267–273, 1979.

[BG54] David Blackwell and M. A. Girshick. *Theory of Games and Statistical Decisions.* John Wiley and Sons, Inc., New York, 1954.

[BL87] James E. Burns and Nancy A. Lynch. The Byzantine firing squad problem. *Advances in Computing Research: Parallel and Distributed Computing*, 4:147–161, 1987. Available as Technical Report MIT/LCS/TM–275, MIT Laboratory for Computer Science.

[Blu] Manuel Blum. Three applications of the oblivious transfer. University of California at Berkeley, 1981.

[BSW69] K. A. Bartlett, R. A. Scantlebury, and P. T. Wilkinson. A note on reliable full-duplex transmission over half-duplex links. *Communications of the ACM*, 12:260–261, 1969.

[CDDS85] Brian Coan, Danny Dolev, Cynthia Dwork, and Larry Stockmeyer. The distributed firing squad problem. In *Proceedings of*

the 17th ACM Symposium on Theory of Computing, pages 335–345, May 1985. Available as IBM Research Report RJ 5343, 1986.

[CL85] K. Mani Chandy and Leslie Lamport. Distributed snapshots: determining global states of distributed systems. *ACM Transactions on Computer Systems*, 3(1):63–75, February 1985.

[CM86] K. Mani Chandy and Jayadev Misra. How processes learn *Distributed Computing*, 1(1):40–52, 1986.

[Coa86] Brian Coan. A communication-efficient canonical form for fault-tolerant distributed protocols. In *Proceedings of the 5th Annual ACM Symposium on Principles of Distributed Computing*, pages 63–72, August 1986.

[DM86] Cynthia Dwork and Yoram Moses. Knowledge and common knowledge in a Byzantine environment I: crash failures (extended abstract). In Joseph Y. Halpern, editor, *Theoretical Aspects of Reasoning about Knowledge: Proceedings of the 1986 Conference*, pages 149–170. Morgan Kaufmann, 1986. To appear in *Information and Computation*. Also available as MIT Technical Memo MIT/LCS/TM-300.

[DRS82] Danny Dolev, Ruediger Reischuk, and H. Raymond Strong. 'Eventual' is earlier than 'Immediate'. In *Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science*, pages 196–203. IEEE, November 1982.

[DS83] Cynthia Dwork and Dale Skeen. The inherent cost of nonblocking commitment. In *Proceedings of the 2nd Annual ACM Symposium on Principles of Distributed Computing*, pages 1–11, 1983.

[DS88] Cynthia Dwork and Larry Stockmeyer. Interactive proof systems with finite state verifiers. Research Report RJ 6262, IBM Almaden Research Center, May 1988.

[ESY84] Shimon Even, Alan L. Selman, and Yacov Yacobi. The complexity of promise problems with applications to public-key cryptography. *Information and Control*, 61:159–173, 1984.

[Fel] Paul Feldman. The optimal prover lives in pspace. Unpublished manuscript.

[FFS87] Uriel Feige, Amos Fiat, and Adi Shamir. Zero knowledge proofs of identity. In *Proceedings of the 19th ACM Symposium on Theory of Computing*, pages 210–217, 1987.

[FH88] Ronald Fagin and Joseph Y. Halpern. Reasoning about knowledge and probability: preliminary report. In Moshe Y. Vardi, editor, *Proceedings of the Second Conference on Theoretical Aspects of Reasoning about Knowledge*, pages 277–293. Morgan Kaufmann, 1988.

[FI86] Michael J. Fischer and Neil Immerman. Foundations of knowledge for distributed systems. In Joseph Y. Halpern, editor, *Theoretical Aspects of Reasoning about Knowledge: Proceedings of the 1986 Conference*, pages 171–186. Morgan Kaufmann, 1986.

[Fis83] Michael J. Fischer. The consensus problem in unreliable distributed systems (a brief survey). Technical Report YALEU/DCS/RR-273, Yale University, June 1983.

[FL82] Michael J. Fischer and Nancy A. Lynch. A lower bound for the time to assure interactive consistency. *Information Processing Letters*, 14(4):183–186, June 1982.

[FMR84] Michael J. Fischer, Silvio Micali, and Charles Rackoff. A secure protocol for the oblivious transfer. In *Eurocrypt*, 1984. This work was presented at the conference, but not published in the proceedings.

[Fre65] J. E. Freund. Puzzle or paradox? *American Statistician*, 19(4):29–44, 1965.

[FZ87] Michael J. Fischer and Lenore D. Zuck. Relative knowledge and belief (extended abstract). Technical Report YALEU/DCS/TR-589, Yale University, December 1987.

[FZ88] Michael J. Fischer and Lenore D. Zuck. Reasoning about uncertainty in fault-tolerant distributed systems. Technical Report YALEU/DCS/TR-643, Yale University, August 1988.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A guide to the Theory of NP-Completeness.* W. H. Freeman and Company, San Francisco, 1979.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, April 1984.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, February 1989.

[GMW86] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity and a methodology of crypotgraphic design. In *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*, 1986.

[Gra78] Jim Gray. Notes on database operating systems. In R. Bayer, R. M. Graham, and G. Seegmuller, editors, *Operating Systems: An Advanced Course*, Lecture Notes in Computer Science, Vol. 66. Springer-Verlag, 1978. Also appears as *IBM Research Report RJ 2188*, Aug., 1987.

[Had83] Vassos Hadzilacos. A lower bound for byzantine agreement with fail-stop processors. Technical Report TR-21-83, Harvard University, 1983.

[Had87] Vassos Hadzilacos. A knowledge-theoretic analysis of atomic commitment protocols. In *Proceedings of the 6th Annual ACM Symposium on Principles of Database Systems*, 1987. Revised version available; submitted for publication.

[Hal50] Paul Halmos. *Measure Theory.* Van Nostrand, 1950.

[HF85] Joseph Y. Halpern and Ronald Fagin. A formal model of knowledge, action, and communication in distributed systems: preliminary report. In *Proceedings of the 4th Annual ACM Symposium on Principles of Distributed Computing*, pages 224–236, 1985.

[HF88] J. Y. Halpern and R. Fagin. Modelling knowledge and action in distributed systems. In F. H. Vogt, editor, *Concurrency 88*, pages 18–32, 1988. A revised and expanded version appears as IBM Research Report RJ 6303, 1988.

[Hin62] J. Hintikka. *Knowledge and Belief.* Cornell University Press, 1962.

[HM84] Joseph Y. Halpern and Yoram Moses. Knowledge and common knowledge in a distributed environment. In *Proceedings of the 3rd Annual ACM Symposium on Principles of Distributed Computing*, pages 50–61, 1984. To appear in JACM. A revised version appears as *IBM Research Report RJ 4421*, Third Revision, September, 1988.

[HM85] Joseph Y. Halpern and Yoram Moses. A guide to the modal logics of knowledge and belief. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, pages 480–490, 1985.

[HMT88] Joseph Y. Halpern, Yoram Moses, and Mark R. Tuttle. A knowledge-based analysis of zero knowledge. In *Proceedings of the 20th ACM Symposium on Theory of Computing*, pages 132–147, May 1988.

[HU85] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley Publishing Company, Reading, Massachusetts, 1985.

[HV89] Joseph Y. Halpern and Moshe Y. Vardi. The complexity of reasoning about knowledge and time, I: Lower bounds. *Journal of Computer and System Sciences*, 38(1):195–237, 1989.

[HZ87] Joseph Y. Halpern and Lenore D. Zuck. A little knowledge goes a long way: Simple knowledge-based derivations and correctness

proofs for a family of protocols. In *Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing*, pages 269–280, August 1987. A revised version appears as *IBM Research Report RJ 5857*, October, 1987.

[Lew80] David Lewis. A subjectivist's guide to objective chance. In W. L. Harper, R. Stalnaker, and G. Pearce, editors, *Ifs*, pages 267–297. D. Reidel Publishing Company, 1980.

[LF82] Leslie Lamport and Michael J. Fischer. Byzantine generals and transaction commit protocols. Technical Report Op. 62, SRI, 1982.

[LS82] D. J. Lehmann and S. Shelah. Reasoning about time and chance. *Information and Control*, 53:165–198, 1982.

[Maz88] Murray S. Mazer. A knowledge theoretic account of recover in distributed systems: The case of negotiated commitment. In *Proceedings of the Second Conference on Theoretical Aspects of Reasoning about Knowledge*, pages 309–324, March 1988.

[Maz89] Murray S. Mazer. *A Knowledge Theoretic Account of Negotiated Commitment.* PhD thesis, Department of Computer Science, University of Toronto, 1989.

[MDH86] Yoram Moses, Danny Dolev, and Joseph Y. Halpern. Cheating husbands and other stories: a case study of knowledge, action, and communication. *Distributed Computing*, 1(3):167–176, August 1986.

[Mic88] Ruben Michel. Efficient protocols for attaining common knowledge and simultaneous byzantine agreement. Technical Report YALEU/DCS/TR-603, Yale University, February 1988.

[Mic89] Ruben Michel. A categorical approach to distrbuted systems expressibility and knowledge. In *Proceedings of the 8th Annual ACM Symposium on Principles of Distributed Computing*, pages 129–144. ACM, August 1989.

[Moo85] Robert C. Moore. A formal theory of knowledge and action. In J. Hobbs and R. C. Moore, editors, *Formal Theories of the Commonsense World.* Ablex Publishing Corp., 1985.

[Mos86] Yoram Moses. *Knowledge in a Distributed Environment.* PhD thesis, Stanford University, March 1986. Available as Stanford University Technical Report STAN-CS-1120.

[Mos88] Yoram Moses. Resource-bounded knowledge. In Moshe Y. Vardi, editor, *Proceedings of the Second Conference on Theoretical Aspects of Reasoning about Knowledge*, pages 261–295. Morgan Kaufmann, March 1988.

[MR89] Yoram Moses and Gil Roth. On reliable message diffusion. In *Proceedings of the 8th Annual ACM Symposium on Principles of Distributed Computing*, pages 119–128. ACM, August 1989.

[MSF83] C. Mohan, H. Raymond Strong, and Shel Finkelstein. Methods for distributed transaction commit and recovery using byzantine agreement within clusters of processors. In *Proceedings of the 2nd Annual ACM Symposium on Principles of Distributed Computing*, pages 89–103. ACM, August 1983.

[MT86] Yoram Moses and Mark R. Tuttle. Programming simultaneous actions using common knowledge. In *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*, pages 208–221. IEEE, October 1986.

[MT88] Yoram Moses and Mark R. Tuttle. Programming simultaneous actions using common knowledge. *Algorithmica*, 3:121–169, 1988.

[NT87] Gil Neiger and Sam Toueg. Substituting for real time and common knowledge in asynchronous distributed systems. In *Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing*, pages 281–293, August 1987. Available as Cornell Technical Report TR 86-790, November, 1986.

[Ore87] Yair Oren. On the cunning power of cheating verifiers: some observations about zero knowledge proofs. In *Proceedings of*

the 28th IEEE Symposium on Foundations of Computer Science, pages 462–471. IEEE, 1987.

[Pag82] H. R. Pagels. *The Cosmic Code: Quantum Mechanics as the Language of Nature.* Simon and Schuster, 1982.

[PR85] R. Parikh and R. Ramanujam. Distributed processes and the logic of knowledge. In *Proceedings of the Workshop on Logics of Programs*, pages 256–268, 1985.

[PSL80] Marshall Pease, Robert Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, 1980.

[PT86] Kenneth J. Perry and Sam Toueg. Distributed agreement in the presence of processor and communication faults. *IEEE Transactions on Software Engineering*, SE-12(3):477–482, 1986.

[PT88] Prakash Panangaden and Kim Taylor. Concurrent common knowledge: A new definition of agreement for asynchronous systems. In *Proceedings of the 7th Annual ACM Symposium on Principles of Distributed Computing*, pages 197–209, August 1988.

[Rab] Michael O. Rabin. Efficient solutions to the distributed firing squad problem. Private communication.

[Rab80] Michael O. Rabin. Probabilistic algorithm for testing primality. *Journal of Number Theory*, 12:128–138, 1980.

[Rab81] Michael O. Rabin. How to exchange secrets by oblivious transfer. Technical Memo TR-81, Aiken Computation Laboratory, Harvard University, 1981.

[Rab82] Michael O. Rabin. N-process mutual exclusion with bounded waiting by $4 \cdot \log n$-valued shared variable. *Journal of Computer and System Sciences*, 25(1):66–75, August 1982.

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public key cryptosystems. *Communications of the ACM*, February 1978.

[SFC85] Dale Skeen, Shel Finkelstein, and Flaviu Cristian. Reliable message diffusion. Unpublished manuscript, 1985.

[Sha85] Glen Shafer. Conditional probability. *International Statistical Review*, 53(3):261–277, 1985.

[Slo89] Robert Sloan. All zero-knowledge proofs are proofs of language membership. Technical Memo MIT/LCS/TM-385, Massachusetts Institute of Technology, February 1989.

[SS77] R. Solovay and V. Strassen. A fast Monte Carlo test for primality. *SIAM Journal on Computing*, 6(1):84–85, March 1977.

[Ste76] M. V. Stenning. A data transfer protocol. *Comput. Networks*, 1:99–110, 1976.

[TW87] Martin Tompa and Heather Woll. Random self-reducibility and zero knowledge interactive proofs of possession of information. In *Proceedings of the 28th IEEE Symposium on Foundations of Computer Science*, pages 472–482, May 1987.

[Var85] Moshe Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *Proceedings of the 26th IEEE Symposium on Foundations of Computer Science*, pages 327–338. IEEE, October 1985.

[vF80] Bas C. van Fraassen. A temporal framework for conditionals and chance. In W. L. Harper, R. Stalnaker, and G. Pearce, editors, *Ifs*, pages 323–340. D. Reidel Publishing Company, 1980.

Index

accepting run, 160
action, 45
action protocol, 23
admissible facts, 84
breaks even, 124
class of system, 44
clients, 40
common knowledge, 25–29, 37
    probabilistic, 135
communication graph, 54
communication tape, 160
communication-efficient protocols, 85
complete test, 175
computation tree, 107
computing a protocol, 85
considers possible, 24–25
consistent probability assignment, 114
corresponding runs, 43
crash failure model, 62
cut, 131
determines, 46, 58
determines safe bets, 125
differing runs, 63
distributed firing squad, 36–37, 46, 81, 83, 87, 96
empty state, 41
everyone knows, 25–28
existence of failures, 46
fact about P, 34
fact about the global state, 34
fact about the initial state, 186
fact about the run, 34
failure models, 41
failure pattern, 42
faulty processor, 42
fixed point axiom, 32
formula, 29
full-information protocol, 49, 56
generalized omissions model, 42, 82–95
generalized omissions with information, 42, 92–93
global state, 21
history, 23
implementable, 46, 96
implements, 46
implicit knowledge, 24–25
inclusive assignments, 118
indexical set, 26
    nonempty, 26
induction rule, 33
initial state, 23
inner measure, 112
input history, 41
input tape, 160
input to a run, 43
interactive proof systems, 161–165
interactive protocols, 159–161
joint view, 24
knowing how to generate, 189–192
knowledge, 24–24
knowledge given facts, 174–178

labeled communication graph, 54
local history, 165
local protocol, 22
local state, 21, 41
measurable sets, 108
measurable with respect to, 116
message history, 41
message protocol, 23
nonfaulty processor, 42
omissions model, 41, 61–79
operating environment, 43
optimal, 36
optimal in all runs, 36
outer measure, 112
perfectly indistinguishable, 166
point, 22
polynomially indistinguishable, 168
practical facts, 58
practical knowledge, 178–180
practical simultaneous choice, 58
practical systems, 58
practically complete tests, 179
practically sound test, 179
probabilistic system, 108
probability assignment, 111
    induced, 116
property, 34
protocol, 22–23
    deterministic, 23
    probabilistic, 23
prover, 159
$\mathcal{P}^{post}$, 119
$\mathcal{P}^{fut}$, 119
$\mathcal{P}^{j}$, 120
$\mathcal{P}^{prior}$, 120
random tape, 160
receiving omissions model, 42, 80–82
rejecting run, 160
resource bounded knowledge, 171–173
resource-bounded knowledge, 97–98
run, 22, 160
run of a protocol, 23
safe bets, 124
sample space assignment, 116
satisfiable, 46
silent processor, 63
similar points, 29
similarity graph, 27–29
simultaneous action, 45
simultaneous Byzantine agreement, 36, 47–48, 64, 76, 78, 83, 87, 93–94, 97–98
simultaneous choice, 45
simultaneous choice problems, 36
simultaneous choice
    strict, 47
sound test, 175
standard assignment, 118
state protocol, 23
state-generated, 34, 113, 115
strategy, 123
strict simultaneous choice, 47
sufficiently rich, 113
system, 22, 43
S5, 31
S^{post}, 119
S^{fut}, 119
S^{j}, 120
S^{prior}, 120
test for common knowledge, 51
transition probability assignment, 108
$Tree_{i,c}$-safe, 124
$Tree^{j}_{i,c}$-safe, 124
uniform assignments, 118
valid, 30
valid at time k, 80
valid in the system, 30
verifier, 159

view, 21
weak interactive proof system, 193–194
weak interactive protocol, 193
work tape, 160
zero knowledge proof systems, 165–169

OFFICIAL DISTRIBUTION LIST

DIRECTOR Information Processing Techniques Office Defense Advanced Research Projects Agency (DARPA) 1400 Wilson Boulevard Arlington, VA 22209	2 copies
OFFICE OF NAVAL RESEARCH 800 North Quincy Street Arlington, VA 22217 Attn: Dr. Gary Koop, Code 433	2 copies
DIRECTOR, CODE 2627 Naval Research Laboratory Washington, DC 20375	6 copies
DEFENSE TECHNICAL INFORMATION CENTER Cameron Station Alexandria, VA 22314	12 copies
NATIONAL SCIENCE FOUNDATION Office of Computing Activities 1800 G. Street, N.W. Washington, DC 20550 Attn: Program Director	2 copies
HEAD, CODE 38 Research Department Naval Weapons Center China Lake, CA 93555	1 copy