Complexity-Theoretic Aspects of Interactive Proof Systems

by

Lance Jeremy Fortnow

B.A., Mathematics and Computer Science
Cornell University
(1985)

Submitted to the Department of Mathematics
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1989

Signature of Author .....
Department of Mathematics
May 5, 1989

Certified by .....
Michael Sipser
Associate Professor, Mathematics
Thesis Supervisor

Accepted by .....
Daniel Kleitman
Chairman, Applied Mathematics Committee

Accepted by .....
Sigurdur Helgason
Chairman, Departmental Graduate Committee

Abstract

In 1985, Goldwasser, Micali and Rackoff formulated interactive proof systems as a tool for developing cryptographic protocols. Indeed, many exciting cryptographic results followed from studying interactive proof systems and the related concept of zero-knowledge. Interactive proof systems also have an important part in complexity theory merging the well established concepts of probabilistic and nondeterministic computation. This thesis will study the complexity of various models of interactive proof systems.

A perfect zero-knowledge interactive protocol convinces a verifier that a string is in a language without revealing any additional knowledge in an information theoretic sense. This thesis will show that for any language that has a perfect zero-knowledge proof system, its complement has a short interactive protocol. This result implies that there are not any perfect zero-knowledge protocols for NP-complete languages unless the polynomial-time hierarchy collapses. Thus knowledge complexity can show a language is easy to prove.

Interesting models of interactive proof systems arise by restricting the power of the verifier. This thesis examines the proof systems with a verifier required to run in logarithmic space as well as polynomial time. Relationships with circuit complexity and log-space Turing machines are developed.

We can increase the power of interactive proof systems by allowing many provers that can not communicate among themselves during the protocol. This thesis shows the equivalence between this multi-prover model and probabilistic Turing machines with an untrustworthy oracle. We additionally give an oracle under which co-NP does not have multi-prover interactive protocols. This result implies an oracle where co-NP does not have standard interactive protocols.

Another natural model occurs when the verifier has only linear time. Towards this direction, this thesis examines probabilistic machines and linear time. We show an oracle under which linear time probabilistic Turing machines can accept all BPP languages, an unusual collapse of a complexity time hierarchy. We exhibit many other related relativized results. Finally we show probabilistic linear time does not contain all languages accepted by interactive proof systems.

Keywords: Computational Complexity, Interactive Proof Systems, Zero-Knowledge, Probabilistic Computation, Oracles

Acknowledgments

Foremost I would like to thank my advisor, Michael Sipser. Mike has collaborated on much of my research and virtually every problem I have ever looked I have discussed with Mike. Mike has greatly influenced my graduate career and I will be forever grateful for these four years I have spent working with him.

I spent the first year of my graduate schooling at the University of California at Berkeley. I started working with Mike Sipser at Berkeley and some of the research in this thesis occurred during that period.

I greatly thank the Office of Naval Research for their generous fellowship that supported me for my first three years in graduate school as well as indirectly covering my travel expenses through out my graduate career. Thanks also to the American Society for Engineering Education for efficiently administering the fellowship.

Thanks to all of the graduate students I have worked and socialized with during these four years. In particular, Eric Schwabe has been a great friend, roommate and proofreader since I arrived at MIT.

Thanks to Marcy Appell for her companionship over the last year and a half and the many new experiences she has given to me (like skiing and coffee).

Finally, thanks to Juris Hartmanis, whose courses at Cornell aroused my interests in theoretical computer science and complexity theory. If Juris had not been at Cornell, I would probably be writing a thesis on Hopf algebras and missing out of the excitement of complexity theory.

Contents

Chapter 1

Introduction

1.1 Victor and the Great Pulu

Victor, a venture capitalist, had everything a man could desire: money, women and power. But he felt something missing. He decided he lacked knowledge. So Victor packed up his bags and headed to the Himalayas in search of ultimate truths.

The natives pointed Victor to a tall mountain and mentioned rumors of a great man full of wisdom. Victor, who smartly brought some climbing equipment, tackled the mountain until he reached a small cave near the summit. Victor found the great Pulu, grand guru of all that is known. Victor inquired to some ultimate truths and Pulu responded,

I will teach you but you must not trust my words.

Victor agreed and found he learned much even though he had to verify all the sayings of the great Pulu. Victor though lacked complete happiness and he asked if he could learn knowledge beyond what he could learn in this manner. The grand guru replied,

You may ask and I will answer.

Victor pondered this idea for a minute and said,

Since you know all that is known, why can you not predict my questions?

A silence reigned over the mountain for a short while until the Guru finally spoke,

You must use other implements—symbols of your past life.

Victor thought for a while and reached into his backpack and brought out some spare change he had unwittingly carried with him. Even the great Pulu can not predict the flip of a coin. He started flipping the coins to ask the guru and wondered *what can I learn now?*

In this thesis we will study the very question of what Victor can learn. On his own Victor can only decide simple problems. With the help of Pulu, even when he can not trust the answers, Victor found he could learn much more than before. With a coin, Victor could learn even more still.

We will formalize these interactions by looking at Victor as a computer with a restricted amount of power. The things Victor can learn on his own form a class of problems, P, containing all the problems a simple computer can solve with this restricted amount of time.

When Victor first meets and listens to Pulu, Victor can now learn problems from the class NP, problems whose answers a simple computer can verify in short amount of time.

Finally when Victor pulls out his coins, we have an interactive proof system with Pulu interacting with Victor, Victor asking random questions to Pulu and Pulu responding in a way to prove certain knowledge to Victor.

Computer scientists do not know if Victor will indeed learn new things by flipping coins with Pulu than on his own though they generally believe he will. We will look at the types of problems Victor can solve with the help of Pulu, concentrating on some variations: what if Pulu wished to reveal no information to Victor beyond the questions Victor asks, what if Victor can communicate with many gurus who can not talk among themselves, and what if we restrict the power of Victor in different ways?

1.2 A Short History

We now go back to the beginning of complexity theory and give a brief history of results leading to the development of the interactive proof system.

Computational complexity theory got its start in 1965 with a paper by Hartmanis and Stearns [HaS] showing simply that if computers have more time they can accept more languages. About the same time, Edmonds [E] introduced the notion that an algorithm runs efficiently if it runs in time polynomial in the size of the input. Cook [C1] defined the class P as the set of all languages with polynomial time algorithms. To this day we use polynomial time as the standard for determining whether a problem has an efficient solution.

Nondeterministic time has its roots in simpler models of computation such as finite and push-down automata. The significance of nondeterminism in computational complexity theory grew in 1971 when Cook [C1] showed a natural problem, satisfiability, is as hard as any other problem in NP, nondeterministic polynomial time. Soon later Karp [Ka] discovered a large number of well known combinatorial problems also had this property. The question of whether polynomial time contains all the languages solvable in nondeterministic polynomial time (P = NP) has become the most famous of open questions in theoretical computer science.

Probabilistic computation came of age in 1977 when Solovay and Strassen [SS] found a probabilistic polynomial-time algorithm to test the primality of a number; a problem still without a provable deterministic polynomial-time solution. Gill [G] defined many of the probabilistic complexity classes including BPP, the problems solvable in probabilistic polynomial time. Probabilistic computation has since played an important role for algorithm designers especially for parallel computer models.

Interactive proof systems, however, owe themselves as much to modern cryptography as to complexity theory. Diffie and Hellman [DH] founded modern cryptography by describing how one might use the hard problems in complexity theory to develop cryptographic protocols. Rivest, Shamir and Adleman [RSA] exhibited a scheme to implement the protocols suggested by Diffie and Hellman. Since then cryptographers have designed many ad hoc protocols for a variety of purposes.

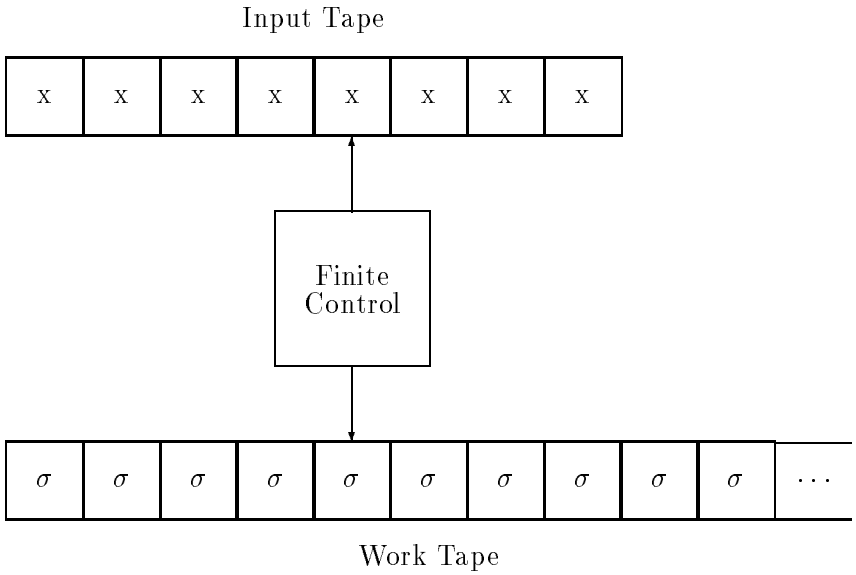

Figure 1-1: A Turing Machine. The x represent the characters of the input string and the σ represent characters on the work tape.

In 1985, Goldwasser, Micali and Rackoff [GMR] developed the interactive proof system as a model for zero-knowledge protocols, protocols that proved the truth of an assumption without revealing any additional information. Also in 1985, Babai [B] introduced Arthur-Merlin games, a variation of the interactive proof system with public randomness. Goldwasser and Sipser [GS] showed the equivalence of these two models.

Zero-knowledge proof systems gained their popularity when Goldreich, Micali and Wigderson [GMW1] showed under certain complexity assumptions that all of NP has zero-knowledge protocols. Many complicated protocols now had trivial reductions to zero-knowledge proof systems.

Interactive proof systems have gained a status as an important complexity class combining both nondeterministic and probabilistic computation. This thesis will concentrate on the complexity aspects of interactive proof systems as opposed to the cryptographic applications.

1.3 Definitions

The basic model of a computer, the *Turing Machine*, consists of a finite state control connected to an input tape and work tape (see figure 1-1). Both the input tape and the work tape consists of cells which can each hold a single letter from a certain finite alphabet Σ. The work tape has an infinite number of cells usable by the Turing machine.

In this thesis we will always assume $\Sigma = \{0, 1\}$ and all logarithms are base two.

The finite state control consists of a set of states including a specified initial state and accepting state, a transition function δ and head pointers to the input tape and the work

tape. The head pointers point to a specific cell on each tape and can be moved right or left. The transition function δ take the values of the cells pointed to by the head pointers and the current state and describes whether to move the head pointers right or left and the new state. The transition function also may specify a change in the contents of the work tape cell pointed to by the work tape head pointer.

The input of a Turing machine consists of *strings*, finite concatenations of letters of the alphabet Σ. The length of a string is the number of letters it contains; there are 2^n strings of length n. We use $|x|$ to represent the length of string x. We let Σ^* designate the set of all possible strings including the zero-length string ϵ. A *language* is a set of strings. A *class* is a set of languages.

Before the Turing machine starts computing, an input string x is placed on the input tape one letter in each cell. The Turing machine initially has pointers to the first cell in the input tape and the first cell on the work tape. The Turing machine computes via the transition function δ in each step moving the head pointers possibly changing the value of the cell in the work tape. We say the Turing machine accepts if it ever enters the accepting state. The Turing machine accepts a language L if it accepts as input strings exactly those strings in L. We let $L(M)$ designate the language accepted by a Turing machine M.

Sometimes we would like a Turing machine to compute a function. We add to the Turing machine model a write only output tape initially blank. When the Turing machine wishes to output a character, the output tape head writes the character and moves one space right. The Turing machine can not perform any other functions on the output tape or head. A Turing machine outputs a string y on input x if that machine outputs exactly the characters of y in order before it halts.

For a more thorough introduction to Turing machines see [HU].

1.3.1 Deterministic Time and Space Complexity

A Turing machine M accepts an input x in time t if M enters an accepting state with at most t applications of the transition function. The machine M accepts a language L in $t(n)$ steps if for all $x \in L$, M accepts x in $t(|x|)$ steps. We will usually use n for $|x|$, the length of the input x.

Suppose we have two functions, $f(n)$ and $g(n)$ from positive integers to positive integers. The function $f(n)$ is $O(g(n))$ if there is some constant c such that $f(n) < cg(n)$ for all n. The function $f(n)$ is $o(g(n))$ if for all constants $c > 0$, $f(n) < cg(n)$ for all but a finite n.

We define the complexity class DTIME$[f(n)]$ as the set of languages accepted by some Turing machine in $O(f(n))$ time. The complexity class P contains all the languages accepted in polynomial time, i.e.

$$\mathrm{P} = \cup_{k>0}\mathrm{DTIME}[n^k]$$

A Turing machine M accepts an input x in space s if M enters an accepting state using only the first s cells of the work tape. In an analogous manner to time, we define the complexity class DSPACE$[f(n)]$. The complexity class PSPACE contains all the languages accepted in polynomial space. The class L contains the languages accepted in logarithmic space, DSPACE$[\log n]$.

1.3.2 Nondeterministic Turing Machines

Nondeterministic computation allows the Turing machine to make guesses. If a series of guesses lead to an accepting state then the Turing machine accepts. Formally, we let the transition function δ have a set of possible moves for a given state. We say the nondeterministic Turing machine M accepts an input string x if there is a choice of transitions that cause M to enter an accepting state.

We define NTIME$[f(n)]$ and NSPACE$[g(n)]$ exactly as DTIME$[f(n)]$ and DSPACE$[g(n)]$ except that the Turing machines involved may be nondeterministic. The classes NL, NP and NPSPACE are the nondeterministic analogues of L, P and PSPACE.

For a complexity class $\mathcal{C}$, we let co-$\mathcal{C}$ contain all languages whose complements belong to $\mathcal{C}$. For example, co-NP contains all languages whose complements are contained in nondeterministic polynomial time. For any two complexity classes, $\mathcal{C}$ and $\mathcal{D}$, if $\mathcal{C} \subseteq \mathcal{D}$ then co-$\mathcal{C} \subseteq$ co-$\mathcal{D}$.

Here are some basic facts relating these complexity classes (see [HU]):

$$\text{DTIME}[f(n)] \subseteq \text{NTIME}[f(n)] \subseteq \text{DTIME}[c^{f(n)}] \text{ (for some } c)$$

$$\text{DSPACE}[f(n)] \subseteq \text{NSPACE}[f(n)] \subseteq \text{DSPACE}[f(n)^2]$$

$$\text{DTIME}[f(n)] \subseteq \text{DSPACE}[f(n)] \subseteq \text{DTIME}[c^{f(n)}] \text{ (for some } c)$$

$$\text{DTIME}[f(n)] = \text{co-DTIME}[f(n)]$$

$$\text{DSPACE}[f(n)] = \text{co-DSPACE}[f(n)]$$

$$\text{NSPACE}[f(n)] = \text{co-NSPACE}[f(n)] \text{ (see [I])}$$

Thus L $\subseteq$ NL $\subseteq$ P $\subseteq$ NP $\subseteq$ PSPACE = NPSPACE and NL = co-NL. None of the inclusions are known to be proper except NL $\neq$ PSPACE.

1.3.3 Completeness

The P = NP question remains the most fundamental open problem in complexity theory. To understand the complexity of NP, we often look at the hardest problems in NP called the NP-complete problems.

Let $f : \Sigma^* \to \Sigma^*$ be a function from strings to strings computable in polynomial time. The function f reduces a language L_1 to a language L_2 if $x \in L_1$ if and only if $f(x) \in L_2$. If L_2 has a polynomial-time solution then L_1 also has a polynomial-time solution for input x by checking if $f(x)$ is in L_2.

A language L is NP-complete if $L \in$ NP and for all languages $L' \in$ NP there is a polynomial-time reduction f from L' to L. Thus if an NP-complete problem has a polynomial-time solution then P = NP. Likewise if P = NP then L has a polynomial-time solution since $L \in$ NP. The P = NP question is equivalent to showing whether any particular NP-complete problem has a polynomial-time solution.

In 1971, Cook [C1] shows the first natural problem, satisfiability, is NP-complete. Satisfiability consists of all boolean formulas such that there exists a setting of the variable to make the setting true. Karp [Ka] shows the NP-completeness of many famous combinatorial

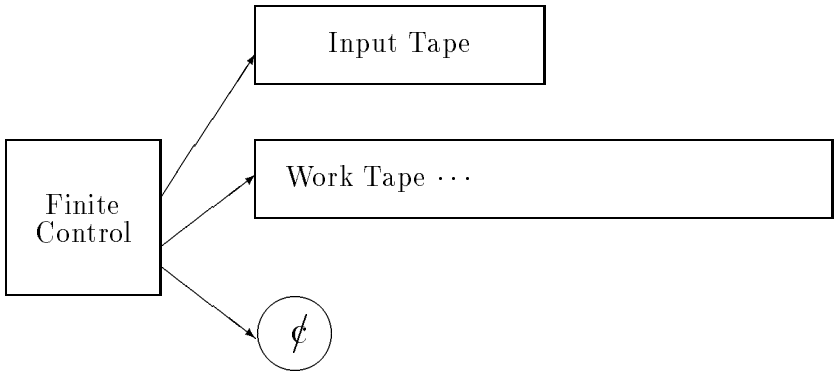

Figure 1-2: A Probabilistic Turing Machine

problems including traveling salesman and vertex cover. Many more results showing various problems NP-complete have since appeared.

For a more in depth discussion of NP-completeness and a list of many of the major NP-complete problems see [GJ].

We can define other forms of completeness. For example we say a language L is log-space complete for P if P contains L and every language in P has a log-space reduction to L.

1.3.4 Probabilistic Turing Machines

Suppose we gave a deterministic Turing machine access to a coin (see figure 1-2). Such a machine could flip the coin and examine different possibilities based on whether the coin came up heads or tails. This machine would accept or reject depending on the outcome of the coin flips.

Formally we define a *probabilistic* Turing machine M by adding a special coin-flip state. When M enters this state it next goes to either a *heads* state or a *tails* state, each with probability one half. Each coin-flip occurs independently of any other coin flip. We can then analyze the probability of M accepting on a certain input.

A probabilistic Turing machine M accepts a language L if for all inputs x in L:

1. If $x \in L$ then $\Pr(M \text{ accepts } x) > 2/3$.
2. If $x \notin L$ then $\Pr(M \text{ accepts } x) < 1/3$.

There is nothing magical about $2/3$ and $1/3$. We can use any two constants strictly between $1/2$ and 1 and strictly between 0 and $1/2$ respectively without affecting the resulting complexity classes.

A proper probabilistic machine M has one-sided error if for $x \notin L$, the probability that M accepts x is zero, *i.e.* M does not accept x on any computation path.

Note that some machines M might not accept any language; for some input x, the probability that M accepts x lies between $1/3$ and $2/3$. We call these *improper* machines.

The language $L \in \text{BPTIME}[f(n)]$ if there is a proper probabilistic machine for L that runs in $O(f(n))$ steps. $\text{BPP} = \cup_{k>0}\text{BPTIME}[n^k]$. We use RTIME and R respectively for

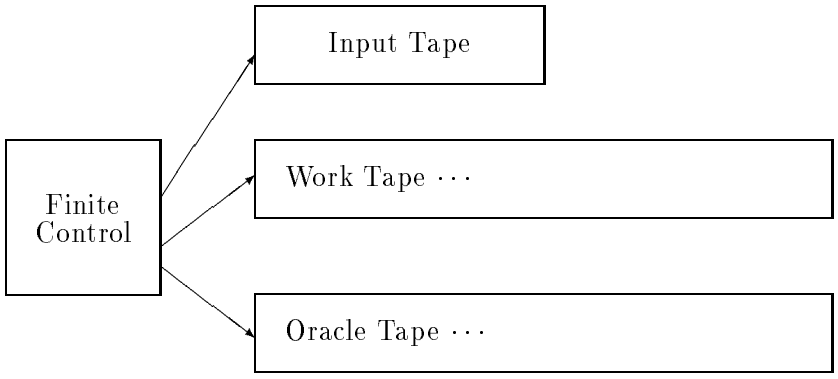

Figure 1-3: A Relativized Turing Machine

one-sided error probabilistic machines. RTIME[$f(n)$] $\subseteq$ NTIME[$f(n)$] because one can just guess the coin tosses. We use ZPTIME and ZPP for zero sided error, *i.e.* a language L has ZPTIME[$f(n)$] machine M if the probabilistic machine M always outputs the correct answer and runs in $O(f(n))$ expected steps. ZPTIME[$f(n)$] = RTIME[$f(n)$]$\cap$co-RTIME[$f(n)$].

The class BPNP combines both nondeterministic and probabilistic computation. The nondeterministic branches choose the path with the best probability of accepting and the probabilistic branches choose each branch with equal probability. BPNP contains all the languages accepted by these machines using the same probabilities as probabilistic machines.

1.3.5 Oracles and The Polynomial Time Hierarchy

Sometimes we would like a Turing machine to have additional information; to have a trustworthy advisor to whom it can ask certain kinds of questions. We will often define an *oracle* that a Turing machine may use.

An oracle A is a subset of Σ^*. A *relativized* Turing machine is a machine with a special oracle tape on which it writes a string $x \in \Sigma^*$ (see figure 1-3). The Turing machine then enters an oracle query state that immediately goes to a special *yes* state if $x \in A$ and to a *no* state otherwise. In shorthand we say the machine makes an oracle *query* of x. We only charge the Turing machine the time it requires to write down the oracle query. We relativize nondeterministic and probabilistic Turing machines in similar ways.

Equivalently, we can think of an oracle as its characteristic function $A : \Sigma^* \rightarrow \{0, 1\}$ where $A(x) = 1$ iff $x \in A$.

We will use superscripts to represent access to an oracle. For example, M^A represents a relativized Turing machine M with access to an oracle A. We can also relativize complexity classes, i.e. the class NP^A consists of all languages recognizable by some polynomial-time nondeterministic Turing machine with access to oracle A. Also we can relativize complexity classes to other complexity classes:

$$\mathrm{P}^{NP} = \cup_{A \in NP} \mathrm{P}^A$$

We relativize complexity classes defined with more than one machine, such as NP$\cap$co-NP, by

allowing every machine access to the same oracle.

If we relativize NP with NP oracles and then relativize this class with NP oracles and so on we get the polynomial-time hierarchy. Formally, we recursively define the hierarchy as follows:

$$\Sigma_0 = \Pi_0 = \Delta_0 = P$$
$$\Sigma_{i+1} = \mathrm{NP}^{\Sigma_i}$$
$$\Pi_{i+1} = \text{co-}\Sigma_{i+1}$$
$$\Delta_{i+1} = \mathrm{P}^{\Sigma_i}$$

The polynomial-time hierarchy, PH, consists of the union of Σ_i for all i.

Some basic facts about the hierarchy (see [St]):

$$\Sigma_1 = \mathrm{NP},\ \Pi_1 = \text{co-NP},\ \Delta_1 = \mathrm{P}$$
$$\mathrm{P} \subseteq \Sigma_1 \subseteq \Sigma_2 \subseteq \cdots \subseteq \mathrm{PH} \subseteq \mathrm{PSPACE}$$
$$\Sigma_i \subseteq \Delta_{i+1} \subseteq \Sigma_{i+1}$$
$$\Pi_i \subseteq \Delta_{i+1} \subseteq \Pi_{i+1}$$

None of the inclusions are known to be proper. We say the polynomial-time hierarchy collapses if $\mathrm{PH} = \Sigma_i$ for some i. Complexity theorists generally believe the hierarchy does not collapse.

Often computer scientists write Σ_i^P for Σ_i to distinguish the polynomial-time hierarchy from the recursion theoretic arithmetic hierarchy. Throughout this thesis Σ_i will refer to the ith level of the polynomial-time hierarchy.

Almost-P contains all the languages accepted by a polynomial-time machine with most oracles. Formally, the class almost-P contains the language L if the set of all R such that $\mathrm{L} \in \mathrm{P}^R$ is measure one in the set of all possible $R \subseteq \Sigma^*$. Often we say L is in P under a random oracle. We can easily show Almost-P = BPP = Almost-BPP. Likewise we can define Almost-NP or for any other complexity class.

1.3.6 Circuits and Nonuniform Computation

Instead of computing via a machine, we can also compute via circuits. A *circuit* consists of *and*, *or* and *not* gates as nodes of a directed graph with the input as the leaves (see figure 1-4). The inputs take on binary values that propagates through the circuit in the obvious manner. The circuit accepts if the value of the root node is one. The size of the circuit equals the number of gates it contains. The depth of the circuit is the length of the longest path. The *fan-in* of a gate g is the number of gates pointing to g.

By using De Morgan's laws (the negation of (x_i *and* x_j) is equivalent to the *or* of the negation of x_i and the negation of x_j) we can assume the *not* gates appear only directly above inputs without increasing the size or depth of the circuit by more than a constant factor.

A family of circuits $\mathcal{C} = \{C_1, C_2, \ldots\}$ consists of a set of circuits where C_n has n inputs. Suppose $x = x_1 x_2 \cdots x_n$ has length n. Then x is accepted by $\mathcal{C}$ if C_n on $x_1, x_2, \ldots, x_n$ accepts. A family of circuits $\mathcal{C}$ accepts a language L if for all x of length n, C_n accepts exactly those x in L.

A family of circuits $\mathcal{C}$ has size $f(n)$ if C_n has size at most $f(n)$ for all n. Similarly we can define depth functions.

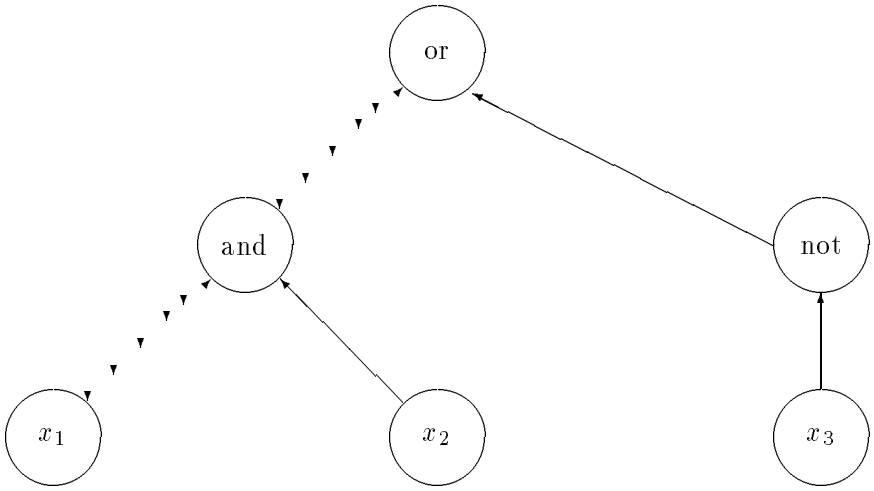

Figure 1-4: A Circuit

There need not be any relationship between the circuits belonging to the same family besides any size and depth restrictions. Often we refer to such circuit families as *nonuniform* circuits.

We relativize a circuit by having a special oracle gate that takes as input a query to the oracle and returns true if that string is in the oracle.

Sometimes we let Turing machines have access to *advice*, a small amount of additional information depending only on the input size. Formally, polynomial advice consists of a list of strings $a_1, a_2, \ldots$ such that $|a_n|$ is bounded by a polynomial in n. A language L is accepted by a polynomial-time Turing machine with polynomial advice if there exists a polynomial-time Turing machine M and polynomial advice such that iff $x \in L$ then M accepts $(x, a_{|x|})$. The class P/poly or *nonuniform* polynomial time consists of all languages accepted in polynomial time with polynomial advice. This class consists of exactly those languages accepted by polynomial size circuits. Likewise we can define NP/poly (nonuniform nondeterministic polynomial time), BPP/log and so on.

Bennet and Gill [BG] have shown all languages in BPP have polynomial size circuits.

Sometimes we require uniformity conditions on our circuits. A circuit family $\mathcal{C}$ is log-space uniform if there exists a log-space Turing machine that outputs C_i on input 1^i. The set of languages accepted by polynomial size log-space uniform circuits consists of exactly those languages in P.

1.3.7 Interactive Proof Systems

An *interactive proof system* consists of two players, an infinitely powerful prover and a probabilistic polynomial-time verifier. The prover will try to convince the verifier of the validity of some statement. However, the verifier does not trust the prover and will only accept if the prover manages to convince the verifier of the validity of the statement.

Formally, an interactive proof system consists of two controls, P, the *prover* and V, the

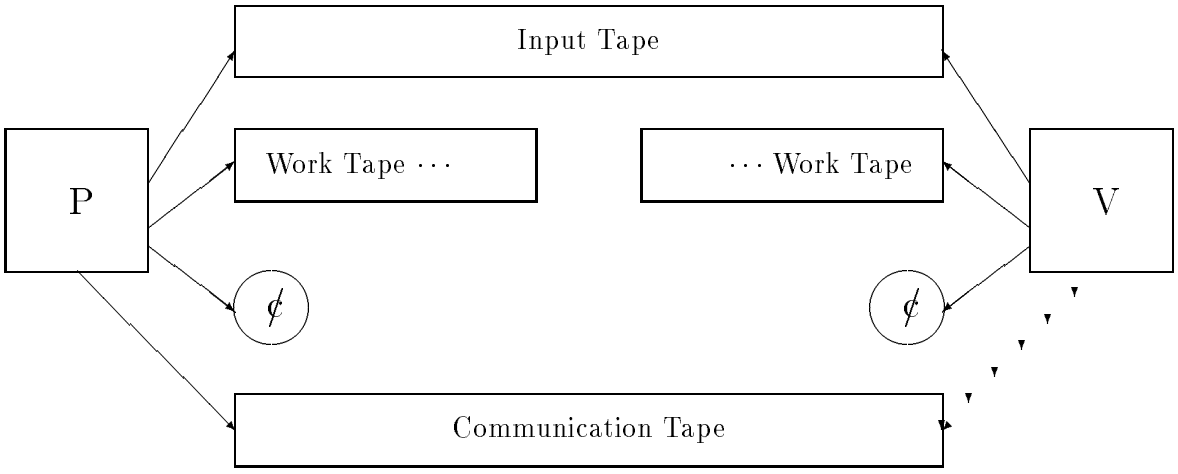

Figure 1-5: An Interactive Proof System

verifier (see figure 1-5). Both the prover and the verifier have access to a common read only input tape and a read/write communication tape. Individually, the prover and verifier have access each to a private coin and a private work tape. The verifier works just as a probabilistic Turing machine. Usually we will restrict the verifier to a polynomial number of steps in the length of the input. However, the prover may compute arbitrary functions of the coin and the contents of the input, work and communication tapes. We put no restrictions on the complexity of this function. Informally we say P has probabilistic infinite power. The proof system accepts if the verifier enters an accepting state. As in the case of probabilistic Turing machines, we are interested in the probability of the verifier accepting.

Communication proceeds by the verifier sending a message to the prover on the communication tape. The size of the messages is limited only in the polynomial time the verifier has to compute. The prover then sends a message to the verifier likewise limited in polynomial size since the verifier has only polynomial time to read it. The prover and verifier may repeat this process a polynomial number of times until the verifier decides whether or not to accept.

An interactive proof system consists of a prover verifier pair $P \leftrightarrow V$. P and V form an interactive protocol for a language L if:

1. If $x \in L$ then $\Pr(P \leftrightarrow V(x) \text{ accepts}) \geq \frac{2}{3}$
2. If $x \notin L$ then for all $P^*, \Pr(P^* \leftrightarrow V(x) \text{ accepts}) \leq \frac{1}{3}$

A *round* of an interactive protocol is a message from the verifier to the prover followed by a message from the prover to the verifier. We let $\mathrm{IP}(f(n))$ represent the languages accepted by interactive proof systems bounded by $f(n)$ rounds. AM is the class of languages accepted by interactive protocols with a fixed constant bound on the number of rounds. Babai [B, BM] shows that one round suffices for AM, *i.e.* $\mathrm{AM} = \mathrm{IP}(1) = \mathrm{IP}(c)$ for any constant c. If we have one round with Merlin (the prover) sending his message before the verifier flips any coins we have the class MA. Babai also shows $\mathrm{MA} \subseteq \mathrm{AM}$.

A *public-coin* interactive proof system allows the prover access to the verifier's coin. Equivalently, we require the verifier's messages to consist of exactly the verifier's coin tosses since the previous round. Goldwasser and Sipser [GS] show the class of languages accepted by the standard interactive proof system is the same as the class of languages accepted by a public coin interactive proof system. This holds for any $f(n)$ bound on the number of rounds.

Messages in a k round conversation will be described by

$$\beta_1, \alpha_1, \beta_2, \ldots, \beta_k, \alpha_k$$

where the α_i are messages from the prover to the verifier at round i and the β_i are messages from the verifier to the prover.

r will be used for the random coin tosses of the verifier.

An interactive proof system runs an *interactive protocol* describing how to perform the computation and communication. The notation for describing protocols follows:

P: These are computations performed by the prover unseen by the verifier. The prover has probabilistic infinite time to make these computations.

$P \rightarrow V$: This is a message from the prover to the verifier.

V: These are computations performed by the verifier unseen by the prover. These computations must be performed in probabilistic polynomial time.

$V \rightarrow P$: This is a message from the verifier to the prover.

Once a protocol has completed, we may wish to execute the protocol again to decrease the amount of error. Each execution should run completely independently. Running an interactive protocol m times in *series* means repeating this process for a total of m times. Running the protocol m times in *parallel* means the verifier sends the first message for all m protocols followed by the prover's responses for all m protocols *et cetera*.

1.4 Relativization

Since the beginning of complexity theory, computer scientists have used techniques from recursion theory to solve computational complexity questions. Often these techniques have been successful; most of the early work in complexity theory has been proven solely using recursion theoretic techniques. However we now know such techniques have limitations. We use oracles to show recursion theoretic techniques can not settle certain complexity questions.

A complexity statement, such as "NP $\subseteq$ PSPACE", is true under an oracle A if the statement is true when all the complexity classes are relativized to the oracle A. We say $\mathrm{NP}^A \subseteq \mathrm{PSPACE}^A$ to mean NP $\subseteq$ PSPACE is true under the oracle A.

When we use recursion theoretic techniques to prove a statement true, the proof will work even if all the machines involved have access to the same oracle. For example, we can prove L $\neq$ PSPACE using diagonalization, i.e. we create a language in PSPACE defined to be different than each possible log-space machine. This proof works even if we allow PSPACE machine and the log-space machines access to any oracle. Thus L $\neq$ PSPACE is true under any oracle A. *All results mentioned in this thesis are true under all oracles unless mentioned otherwise.* We say a technique relativizes if its application is independent of any oracle access.

Suppose we show a complexity statement is true under a certain oracle B. If we could prove the statement false using recursion theoretic methods then the statement would be false under all oracles. This contradiction tells us recursion theoretic techniques will not work to prove the statement false. If we can find two oracles A and B such that the statement is true under oracle A and false under oracle B then recursion theoretic techniques will not work to settle the statement true or false. We will need other techniques, techniques that do not relativize, to settle this statement. Most complexity statements relativized both true and false have remained unsettled by any techniques.

In 1975, Baker, Gill and Solovay [BGS] had the first use of oracles to show the famous P = NP? question likely has a hard solution. They have found oracles A and B such that $\mathrm{P}^A = \mathrm{NP}^A$ and $\mathrm{P}^B \neq \mathrm{NP}^B$. Thus techniques that relativize will not settle the P = NP? question. They also show the existence of oracles C, D, E and F such that:

$$\mathrm{NP}^C \neq \text{co-}\mathrm{NP}^C$$
$$\mathrm{P}^D \neq \mathrm{NP}^D = \text{co-}\mathrm{NP}^D$$
$$\mathrm{P}^E \neq \mathrm{NP}^E \text{ and } \mathrm{P}^E = \mathrm{NP}^E \cap \text{co-}\mathrm{NP}^E$$
$$\mathrm{P}^F \neq \mathrm{NP}^F \cap \text{co-}\mathrm{NP}^F \text{ and } \mathrm{NP}^F \neq \text{co-}\mathrm{NP}^F$$

Baker, Gill and Solovay left open some questions about the polynomial-time hierarchy that remained unsolved for over a decade. In 1985, Yao [Y] showed an oracle A such that the polynomial-time hierarchy did not collapse under A, *i.e.* $\Sigma_i^A \neq \Sigma_{i+1}^A$ for all i. In 1988, Ko [Ko] found a series of oracles $A_1, A_2, \ldots$ such that for each i, the polynomial-time hierarchy collapsed to the ith level under oracle A_i, i.e. $\Sigma_{i-1}^{A_i} \neq \Sigma_i^{A_i} = \Sigma_{i+1}^{A_i} = \mathrm{PH}^{A_i}$.

Rackoff [R] shows some relativized results about probabilistic complexity classes. Rackoff found oracles A and B such that $\mathrm{P}^A = \mathrm{R}^A \neq \mathrm{NP}^A$ and $\mathrm{P}^B \neq \mathrm{R}^B = \mathrm{NP}^B$. For example, we could not easily prove P $\neq$ R even if we assume P $\neq$ NP.

This section gives just a small sample of the many oracle results proven over the last fifteen years. Via oracle results, we can learn what we will have difficulty proving; particularly which techniques will not work. This thesis will have several examples of relativized results to show which questions of interactive proof systems may be hard to resolve.

1.5 Basic Results

We have come a long way in understand the complexity of interactive proof systems since Goldwasser, Micali and Rackoff [GMR] developed them in 1985. This section will give a review of important complexity results not covered in this thesis.

In 1986, Goldwasser and Sipser [GS] showed private-coin interactive proof systems and public-coin interactive proof systems accepted exactly the same class of languages. In fact the class of languages accepted $f(n)$-round private-coin interactive proof systems contains exactly the same languages accepted by $f(n)$-round public-coin interactive proof systems.

At the same time as interactive proof systems Babai [B] invented his Arthur-Merlin games similar to interactive proof systems but using public coins for the verifier instead of private coins. Goldwasser and Sipser show the equivalence between these two models.

Babai [B, BM] showed a one-round proof system can accept all the languages of any constant-round proof system. More generally, he shows $\mathrm{IP}(cf(n)) = \mathrm{IP}(f(n))$ for any constant c. In other words we can reduce the number of rounds of an interactive protocol by any

constant factor. We can not easily improve this result because Aiello, Goldwasser and Hastad show, for any functions $f(n)$ and $g(n)$ with $f(n) = o(g(n))$, the existence of an oracle A such that $\mathrm{IP}(f(n)) \neq \mathrm{IP}(g(n))$. In particular they exhibit an oracle that separates AM from IP.

Babai also showed MA $\subseteq$ AM, though Zachos [Z] also proved the same result using his generalized quantifier swapping techniques. Santha [Sa] exhibits an oracle that separates MA from AM.

Babai and Moran [BM] show how to decrease the error probability without increasing the number of rounds by running the protocol several times in parallel. If $x \in L$ then we can make the probability of acceptance at least $1 - 2^{-p(n)}$ and for $x \notin L$ we can make the probability of acceptance at most $2^{-p(n)}$ for any polynomial $p(n)$.

Goldreich, Mansour and Sipser [GMS] show any language that has a $f(n)$-round interactive protocol has a $f(n)+1$-round interactive protocol such that if $x \in L$ then the verifier always accepts. Note that this is opposite to the notion of one-sided error used to define the class R. Goldreich, Mansour and Sipser also show only NP languages have interactive proofs such that the verifier always rejects if $x \notin L$.

How do the interactive proof classes compare to the polynomial-time hierarchy? Sipser and Gács [Si] showed that BPP $\subseteq \Sigma_2 \cap \Pi_2$. An elegant proof of this fact by Lautemann [La] easily generalizes to show MA $\subseteq \Sigma_2 \cap \Pi_2$ and AM $\subseteq \Pi_2$ [BM]. Santha's oracle [Sa] actually shows a language in AM but not in Σ_2 under that oracle. Feldman [Fe] shows PSPACE contains IP though the oracle created by Aiello, Goldwasser and Hastad [AGH] puts IP outside of the polynomial-time hierarchy.

Fortnow and Sipser [FS1] exhibit an oracle such that IP does not contain co-NP. A generalized version of this proof appears in section 4.4. Boppana, Hastad and Zachos [BHZ] show that if AM contains co-NP then the polynomial hierarchy collapses to Σ_2.

Nisan and Wigderson [NW] show the equivalence between AM and almost-NP. Goldwasser and Sipser [GS] note the equivalence of IP and BPNP and that nonuniform NP contains all of AM.

1.6 An Example: Graph Nonisomorphism

To help in the understanding of interactive proof systems, we will now examine in detail an interactive protocol for graph nonisomorphism developed by Goldreich, Micali and Wigderson [GMW1].

An undirected graph $G = (V, E)$ consists of a set of vertices $V = \{v_1, v_2, \ldots\}$ and a set of edges E of unordered pairs of vertices. Let π be a permutation of the first n integers. We define $\pi(G)$ as the graph obtained by applying the permutation π to the graph G, *i.e.* the edges of $\pi(G)$ consist of all edges $(\pi(u), \pi(v))$ such that (u, v) are edges of G.

The two graphs G_1 and G_2 are isomorphic if there exists a permutation π such that $G_1 = \pi(G_2)$. Let GI be the language consisting of pairs of graphs (G_1, G_2) such that G_1 and G_2 are isomorphic. A nondeterministic polynomial-time machine can determine whether there exists an isomorphism between G_1 and G_2 by guessing a permutation π and verifying that $G_1 = \pi(G_2)$. Let GNI consist of the pairs of nonisomorphic graphs.

Since GI $\in$ NP then GI has a trivial interactive protocol by the prover sending to the verifier the permutation π. GI is not known to be in P or NP-complete. GNI is not known to be in NP; however we will exhibit an interactive protocol for graph nonisomorphism.

Let G_1 and G_2 be two graphs each with n vertices. We create a prover, P, and a verifier, V to run the following protocol:

V: Pick a permutation π at random and pick $i \in \{1, 2\}$ also at random. Compute $G = \pi(G_i)$.

$V \rightarrow P$: G

$P \rightarrow V$: j

V: Accept if $j = i$.

Suppose G_1 and G_2 were not isomorphic. Then $\pi(G_1)$ could only be isomorphic to G_1 and not G_2 and vice versa. An infinitely powerful prover could identify which graph was isomorphic to G and correctly respond with a j such that $j = i$.

Now suppose G_1 was isomorphic to G_2. Then G will be isomorphic to both G_1 and G_2. Despite its infinite power, any prover just will not have enough information to determine the graph is isomorphic to G.

If the language GNI contains (G_1, G_2) then the protocol will cause the verifier to accept with probability one. If however the graphs are isomorphic the protocol accepts with probability at most one half. Unfortunately we require in the definition of interactive proof systems that the proof system accept with probability at most one third for strings (G_1, G_2) not in the language GNI.

We solve this problem by running the protocol two times in series and having the verifier accept if both times the prover correctly determined the original graph. For nonisomorphic graphs the probability of acceptance remains at one but for isomorphic graphs the prover can correctly guess the original graph both times with probability at most one fourth. Thus we now have a three-round interactive protocol for graph nonisomorphism.

Goldwasser and Sipser [GS] show us how to convert this interactive protocol into a public coin protocol; a surprising result because the protocol above depends on the verifier keeping the choice of i a secret. Since we have a bounded-round protocol, Babai [B, BM] shows us how to convert this protocol into a two round AM protocol where the verifier sends random public coins followed by the prover's response. Thus GNI $\subseteq$ almost-NP and GNI $\subseteq$ NP/poly though graph nonisomorphism is not known to be in NP or BPP. If GI was NP-complete then the polynomial-time hierarchy would collapse to Σ_2.

We will return to graph isomorphism in section 2.5.2.

Chapter 2

Perfect Zero-Knowledge

2.1 A Cryptographic Side of Interactive Proof Systems

When Goldwasser, Micali and Rackoff [GMR] developed interactive proof systems, they concurrently developed zero-knowledge, a restriction of interactive proof systems requiring the verifier not learn any additional knowledge useful to him as a polynomial-time machine. Goldreich, Micali and Wigderson [GMW1] show if one way functions exist then all languages in NP have zero-knowledge proofs. However, their proof relies on the fact that the verifier has limited power and is unable to invert these one-way functions. Perfect zero-knowledge (PZK), a stronger restriction, requires the verifier not learn any additional information no matter how powerful he may be. There are several languages not known to be in BPP or NP$\cap$co-NP, such as graph isomorphism [GMW1], which have perfect zero-knowledge proof systems.

Our main theorem shows for any language that has a perfect zero-knowledge proof system, its complement has a single-round interactive proof system. Thus PZK $\subseteq$ co-AM, the complement of languages accepted by one-round interactive proof systems. Our result holds in the weaker case where we only require the verifier following the protocol will not learn any additional information.

Combining our main theorem with a result of Boppana, Hastad and Zachos [BHZ], we show NP-complete languages do not have perfect zero-knowledge proof systems unless the polynomial-time hierarchy collapses to the second level. Thus it is unlikely that the result of Goldreich, Micali and Wigderson will extend to perfect zero-knowledge.

2.2 Notation and Definitions

Let $P \leftrightarrow V$ represent an interactive protocol between a prover P and a verifier V. The verifier's *view of the conversation* consists of all the messages between P and V and the random coin tosses of V.

Let M be a simulator for a view of the conversation between P and V. The simulator M is a probabilistic polynomial-time machine that will output a conversation between P and V including the random coin tosses r of V. Thus each run of M will produce:

$$r, \beta_1, \alpha_1, \beta_2, \ldots, \beta_k, \alpha_k$$

where the α_i are messages from the prover to the verifier at round i and the β_i are messages from the verifier to the prover.

Let $P \leftrightarrow V[x]$ denote the probability distribution of views of conversations between P and V. Let $M[x]$ denote the distribution of views of conversations created by running M on x.

Let $A[x]$ and $B[x]$ be two distributions of strings. The distributions $A[x]$ and $B[x]$ are *statistically close* if for any subset of strings $\mathcal{S}$,

$$\left| \sum_{y \in S} \Pr_{A[x]}(y) - \sum_{y \in S} \Pr_{B[x]}(y) \right| < \frac{1}{q(|x|)}$$

for all polynomials q with $|x|$ large enough. Let J be a probabilistic polynomial-time machine that outputs either 0 or 1. The distributions $A[x]$ and $B[x]$ are *polynomial-time indistinguishable* if for any J,

$$|\Pr(J(A[x]) = 1) - \Pr(J(B[x]) = 1)| < \frac{1}{r(|x|)}$$

for all polynomials r with $|x|$ large enough. Let $J(A[x])$ be the output of J when run on a string chosen from the probability distribution $A[x]$. Note if $A[x]$ and $B[x]$ are statistically close then they are polynomial-time indistinguishable.

$P \leftrightarrow V$ is *(computational) Zero-Knowledge* (ZK) if for any verifier V^* there is a M_{V^*} such that for all x in L, $P \leftrightarrow V^*[x]$ and $M_{V^*}[x]$ are polynomial-time indistinguishable. We use zero-knowledge throughout this thesis to refer to computational zero-knowledge.

$P \leftrightarrow V$ is *Perfect Zero-Knowledge* (PZK) if for any verifier V^* there is a M_{V^*} such that for all x in L, $P \leftrightarrow V^*[x] = M_{V^*}[x]$.

$P \leftrightarrow V$ is *Statistical Zero-Knowledge* (SZK) if for any verifier V^* there is a M_{V^*} such that for all x in L, $P \leftrightarrow V^*[x]$ and $M_{V^*}[x]$ are statistically close.

Goldwasser, Micali and Rackoff [GMR] introduced zero-knowledge as well as interactive proof systems in 1985.

Note ZK $\supseteq$ SZK $\supseteq$ PZK. The inclusions are not known to be proper but the main result of this chapter gives good evidence that ZK $\neq$ SZK.

The results in this chapter only require a weaker version of zero-knowledge: a simulator only need exist for the given P and V and not necessarily for any V^*. For the rest of this chapter we will assume this weaker model and use M for M_V, the simulator for P and V.

2.3 Related Results

Goldreich, Micali and Wigderson [GMW1] show every language in NP has a zero-knowledge interactive proof system if a one-way function exists. This result does not relativize; there exists an oracle such that one way functions exist but NP does not have zero-knowledge proofs. The proof of Goldreich, Micali and Wigderson works by exhibiting a zero-knowledge proof for a certain NP-complete problem instead of for general nondeterministic machines and thus the proof does not relativize.

Our result shows for any language L with an statistical zero-knowledge proof system, there exists a bounded-round interactive proof system for its complement $\overline{L}$. We can then apply several earlier results about bounded-round interactive proof systems described in section 1.5.

Subsequent to our result, Aiello and Hastad [AH] have shown, using similar techniques, a bounded-round interactive proof system can simulate any statistical zero-knowledge proof system. This nice complement to our result combines with our result to show nonuniform NP∩co-NP contains any language with a perfect zero-knowledge proof system.

Brassard and Crépeau [BC] have shown perfect zero-knowledge for SAT using a different model for interactive proof systems where the prover is a polynomial-time machine that knows a satisfying assignment. Our result about perfect zero-knowledge relies on the ability of the prover to have infinite power and does not apply to Brassard and Crépeau's model.

2.4 Showing Sets are Large and Small

In this chapter, we will need protocols to show sets are large and small. We create both protocols using Carter-Wegman Universal Hash Functions [CW].

Suppose $\mathcal{S} \subseteq \Sigma^N - \{0^N\}$. For F a random binary $b \times N$ matrix, let $f : \Sigma^N \to \Sigma^b$ be the function defined by $f(x) = Fx$ using regular matrix multiplication modulo two. We can think of f in terms of linear algebra over the field of two elements. The distribution of f forms the uniform distribution over all possible linear functions from n-dimensional space to b-dimensional space. Let $f_{\mathcal{S}}$ be the function f restricted to the domain $\mathcal{S}$.

If $|\mathcal{S}| \gg 2^b$ then $f_{\mathcal{S}}$ is likely to be onto most of Σ^b and most elements of Σ^b will have many preimages.

If $|\mathcal{S}| \ll 2^b$ then the range of $f_{\mathcal{S}}$ is a small subset of Σ^b and most elements of $f_{\mathcal{S}}(\mathcal{S})$ will have only one inverse in $\mathcal{S}$.

Lemma 2.1 (Vector Independence) *Suppose $x_1, x_2, \ldots, x_k \in \Sigma^N$ are linearly independent vectors over the field of two elements. Then $f(x_1), f(x_2), \ldots, f(x_k)$ are independently and uniformly distributed over Σ^b.*

Proof Since $x_1, x_2, \ldots, x_k$ are linearly independent, we can extend to a basis. Let T be the transformation matrix from this new basis to the canonical basis of Σ^N. Then the matrix $B = FT$ describes the function from the new basis to the canonical basis of Σ^b. Since T is an invertible matrix, there is a one-to-one correspondence between B and F. Thus B is distributed uniformly over all possible binary $b \times N$ matrices. The vector $f(x_j)$ is just the jth column of B. Thus each $f(x_j)$ is independently distributed over Σ^b. □

2.4.1 Lower Bound Protocol

Goldwasser and Sipser [GS] developed the following protocol to show $\mathcal{S}$ is large for $\mathcal{S}$ recognizable in polynomial time:

V: Pick ℓ independent random hash functions $f_1, \ldots, f_\ell : \Sigma^N \to \Sigma^b$ and ℓ^2 points $z_1, \ldots, z_{\ell^2} \in \Sigma^b$

$V \to P$: $f_1, \ldots, f_\ell, z_1, \ldots, z_{\ell^2}$

$P \to V$: x

V: Accept if $x \in \mathcal{S}$ and $f_i(x) = z_j$ for some i, j, $1 \leq i \leq \ell$ and $1 \leq j \leq \ell^2$

If $\mathcal{S}$ is much smaller than 2^b then there will probably be no x such that $f_i(x) = z_j$. However if $\mathcal{S}$ is large then there will probably be many x fulfilling $f_i(x) = z_j$ and a infinitely powerful prover will have no trouble exhibiting such an x that V can verify in polynomial time.

Lemma 2.2 (Lower Bound) [GS] *Using the above protocol with a given $N, b, d > 0$ and $\ell > \max\{b, 8\}$*

1. *If $|\mathcal{S}| \geq 2^b$ then $\Pr(P \leftrightarrow V \text{ accepts}) \geq 1 - 2^{-\frac{\ell}{8}}$*
2. *If $|\mathcal{S}| \leq \frac{2^b}{d}$ then $\Pr(P^* \leftrightarrow V \text{ accepts}) \leq \frac{\ell^3}{d}$ for any P^**

2.4.2 Upper Bound Protocol

If V has a random element s in $\mathcal{S}$ completely unknown to P then we can use the following protocol to show $\mathcal{S}$ is small:

V: Pick a random $N \times b$ matrix F

$V \rightarrow P$: $F, f(s) = Fs$

$P \rightarrow V$: s

For small $\mathcal{S}$, it is unlikely more than one element of $\mathcal{S}$ will map to $f(s)$ and P can determine s. For large $\mathcal{S}$, probably many s will map to $f(s)$ and P can not determine which element of $\mathcal{S}$ the verifier had.

Lemma 2.3 (Upper Bound) *Using the above protocol with a given $N, b > 0$ and $d > 2$*

1. *If $|\mathcal{S}| \leq \frac{2^b}{d}$ then $\Pr(P \leftrightarrow V \text{ accepts}) \geq 1 - \frac{1}{d}$*
2. *If $|\mathcal{S}| > 8d2^b$ then $\Pr(P^* \leftrightarrow V \text{ accepts}) \leq \frac{1}{d}$ for any P^**

Proof Let A be the random variable equal to the number of $x \neq s$ in $\mathcal{S}$ such that $f(x) = f(s)$. Let $\mathcal{S}' = \mathcal{S} - \{s\}$. Let A_x be the indicator random variable equal to one if $f(x) = f(s)$, zero otherwise. Then

$$E(A) = E(\sum_{x \in \mathcal{S}'} A_x) = \sum_{x \in \mathcal{S}'} E(A_x) = \sum_{x \in \mathcal{S}'} 2^{-b} = \frac{|\mathcal{S}| - 1}{2^b}$$

If $|\mathcal{S}| \leq \frac{2^b}{d}$ then $E(A) \leq \frac{1}{d}$. If $f(s)$ has only s as an inverse in $|\mathcal{S}|$ then P with his infinite power will be able to determine s. Thus $\Pr(P \leftrightarrow V \text{ rejects}) \leq \Pr(A \geq 1) \leq E(A) \leq \frac{1}{d}$ since A is an integral random variable.

Suppose $|\mathcal{S}| > 8d2^b$. We can assume $|\mathcal{S}| = 8d2^b + 1$ without increasing the probability of acceptance. Then $E(A) = 8d$. Since P^* has no idea what s the verifier V has, P^* can only have a $\frac{1}{A+1}$ probability of predicting the s that V has. We will show with a large probability A is large using the variance of A.

Given x, y, s all distinct and $y \neq x \oplus s$ then x, y and z are linearly independent. Then by the Vector Independence Lemma $f(x)$, $f(y)$ and $f(s)$ are independently distributed over

Σ^b. It then follows that A_x and A_y are independent random variables and their covariance is zero.

The covariance of any two indicator random variables is never greater then the expected value of one of them. Then VAR(A) =

$$\sum_{x,y\in\mathcal{S}'} COV(A_x, A_y) = \sum_{x\in\mathcal{S}'} (COV(A_x, A_x) + COV(A_x, A_{x\oplus s})) \leq \sum_{x\in\mathcal{S}'} 2E(A_x) \leq 16d$$

Possibly $x \oplus s \notin \mathcal{S}$ which could only decrease the variance. Using Chebyshev's inequality we get:

$$\Pr(A < 2d) \leq \Pr(|A - 8d| \geq 6d) \leq \frac{VAR(A)}{36d^2} \leq \frac{16d}{36d^2} \leq \frac{1}{2d}$$

So with probability at most $\frac{1}{2d}$, the prover P^* can determine s easily because A is small enough; otherwise P^* has at most $\frac{1}{2d}$ chance of guessing s, so in total P^* has at most a $\frac{1}{d}$ chance of determining s. □

2.4.3 Comparison Protocol

Suppose we had two sets $\mathcal{S}_1, \mathcal{S}_2 \subseteq \Sigma^N$ and wanted to show $|\mathcal{S}_1| \gg |\mathcal{S}_2|$. If $\mathcal{S}_1$ is polynomial-time testable and V has a random element s_2 of $\mathcal{S}_2$ then we can use the following protocol to show $|\mathcal{S}_1| \gg |\mathcal{S}_2|$:

$P \rightarrow V$: $b' \leq N$

$P \rightarrow V$: Use lower-bound protocol on $\mathcal{S}_1$ with $b = b', \ell = 8nN$

$P \rightarrow V$: Use upper-bound protocol on $\mathcal{S}_2$ with $b = b' - 3n, s = s_2$

V: Accept if both the upper and lower bound protocols accept

Lemma 2.4 (Comparison) *Using the above protocol*

1. *If* $|\mathcal{S}_1| \geq 2^{4n+1}|\mathcal{S}_2|$ *then* $\Pr(P \leftrightarrow V$ *accepts*$) \geq 1 - 2^{1-n}$
2. *If* $|\mathcal{S}_1| \leq 2^{n-4}|\mathcal{S}_2|$ *then* $\Pr(P^* \leftrightarrow V$ *accepts*$) \leq n^3N^3 2^{9-n}$ *for any* P^*

Proof Let $d = 2^n$.

1. Pick $b' = \lfloor \log |\mathcal{S}_1| \rfloor$. Then each protocol accepts with probability $\geq 1 - 2^{-n}$, so both will accept with probability $\geq 1 - 2^{1-n}$ by the upper and lower bound lemmas.
2. There are two cases depending on what b' the prover P chooses
   (a) If $b' \geq \lceil \log |\mathcal{S}_1| \rceil - n$ then by the lower bound lemma the probability of V accepting is $\leq n^3N^3 2^{9-n}$
   (b) If $b' < \lceil \log |\mathcal{S}_1| \rceil + n$ then by the hypothesis $b' - 3n < \lfloor \log |\mathcal{S}_2| \rfloor - n - 3$ and by the upper-bound lemma the probability of V accepting is $\leq 2^{-n}$. □

Using Carter-Wegman Hashing to show a set is large was introduced by Sipser [Si] and used extensively in [Si, B, GS]. To the author's knowledge this is the first use of an interactive protocol to show a set is small.

2.5 Main Theorem

We will start with a simple version of the theorem:

Theorem 2.5 *For any language L with a perfect zero-knowledge interactive proof system there exists an interactive proof system accepting $\overline{L}$.*

2.5.1 Structure of Proof

We are given a prover and verifier (P and V) for the language L, and a simulator M that produces views of conversations between P and V and the random coin tosses of V. A probabilistic polynomial-time machine can simulate the computation of V checking, for example, whether or not V accepts. On $x \in L$, M produces a view of a conversation from exactly the same probability distribution as when P and V run on x. However, the definition of perfect zero-knowledge has no requirements on the simulator in the case when $x \notin L$; three possibilities arise:

1. M will produce "garbage", something clearly not a randomly selected member of $P \leftrightarrow V[x]$.
2. M will produce views of conversations that cause V to reject most of the time.
3. M will produce a simulation that looks valid and causes V to accept. It may not be possible in polynomial time to differentiate this view from one created by P and V when $x \in L$. However, M must produce views of conversations from a distribution quite different from the distribution of views between P and V, since in the real views V will probably reject.

We will create a new prover and verifier, P' and V', to determine if one of these three cases occur. The verifier V' will simulate M and get a view of a conversation between P and V as well as r, the random coin tosses of V. The verifier V' will check the validity of this view and that V accepts. If the view fails this test then it fails in cases 1 or 2 so V' accepts knowing $x \notin L$. Otherwise V' will send to P' some initial segment of the conversation. The prover P' will then convince V' that the conversation came from a bad distribution by "predicting" r better than P' could have done from a good distribution.

2.5.2 An Example: Graph Isomorphism

In section 1.6, we discussed graph isomorphism and showed an interactive protocol for graph nonisomorphism. In this section, we will present a perfect zero-knowledge protocol for graph isomorphism developed by Goldreich, Micali and Wigderson [GMW1] and show how our theorem converts this zero-knowledge protocol to an interactive protocol for graph nonisomorphism. The protocol we develop will be virtually identical to the protocol for graph nonisomorphism in [GMW1] and discussed in section 1.6; our proof, however, shows the similarity between the two protocols is not coincidental.

Recall two graphs G_1 and G_2 are isomorphic if there exists a permutation π such that $G_2 = \pi(G_1)$. A perfect zero-knowledge protocol for graph isomorphism suggested by [GMW1] works as follows:

P: Generate random permutation π and computes $G = \pi(G_1)$

$P \rightarrow V$: G

$V \rightarrow P$: $i = 1$ or 2 chosen at random

$P \rightarrow V$: π' chosen at random such that $\pi'(G_i) = G$

If $G_1 \cong G_2$ then G will be a permutation of both G_1 and G_2 and P will always be able to find a π'. If $G_1 \ncong G_2$ then G cannot be a permutation of both G_1 and G_2, so at least half of the time V will choose an i such that no π' exists.

Thus we have an interactive protocol for graph isomorphism. This protocol also is perfect zero-knowledge. The simulator M works as follows:

M generates π and i at random and computes $G = \pi(G_i)$, then outputs the following view of a conversation:

r: i

$P \rightarrow V$: G

$V \rightarrow P$: i

$P \rightarrow V$: π

It is easy to verify when $G_1 \cong G_2$, M produces exactly the same distribution of views of conversations as P and V. Notice what happens when $G_1 \ncong G_2$. The output of M always causes V to accept. Thus when $G_1 \ncong G_2$, M must produce views of conversations from a very different distribution from what P and V produce. In fact whenever $G_1 \ncong G_2$, one can always predict $r = i$ from the G produced by M.

This leads to a new interactive protocol between a new prover and verifier, P' and V', for graph nonisomorphism as follows:

V': Generate π and i at random and compute $G = \pi(G_i)$

$V' \rightarrow P'$: G

$P' \rightarrow V'$: j

V': Accept if $j = i$

2.5.3 The Protocol

We have a prover and verifier, P and V for a language L and a simulator M such that M exactly simulates views of conversations between P and V for x in L. Let $n = |x|$ and let k be the number of rounds of the protocol (bounded by a polynomial in n). We can decrease the probability of error in the protocol between P and V to $2^{-p(n)}$ for any polynomial $p(n)$ by the standard trick of running the protocol several times in parallel and having V accept if the majority of individual protocols accept [BM]. This new protocol is still perfect zero-knowledge—we just run the simulator in parallel. Note we make use of the fact that we only need a simulator for the real verifier V. In general, it is not known whether perfect zero-knowledge protocols remain perfect zero-knowledge when run in parallel.

Thus we can assume:

1. If $x \in L$ then $\Pr(P \leftrightarrow V(x) \text{ accepts}) \geq 1 - 2^{-6kn}$
2. If $x \notin L$ then for all P^*, $\Pr(P^* \leftrightarrow V(x) \text{ accepts}) \leq 2^{-6kn}$

For the sake of the comparison protocol, we require V immediately reject if all its coin tosses are zero. Since this will happen with an exponentially small probability it will not affect the correctness of the protocol. The protocol remains perfect zero-knowledge by having the simulator output no conversation if the verifier's coins are all zero.

A protocol between a new prover and verifier, P' and V', works as follows:

V': Run M and get $r, \beta_1, \alpha_1, \ldots, \beta_k, \alpha_k$. The verifier V' now checks:

1. the validity of the conversation, *i.e.* $r, \alpha_1, \ldots, \alpha_k$ will cause V to say $\beta_1, \ldots, \beta_k$.
2. the conversation causes V to accept.

If either of these tests fail then V' can be very sure that $x \notin L$ so V' quits now and accepts. Otherwise V' continues.
Let $j = 1$.

$V' \rightarrow P'$: β_j, α_j

$P' \rightarrow V'$: Look at the sets $\mathcal{R}_1$ and $\mathcal{R}_2$ as defined below. If $|\mathcal{R}_1| \geq 2^{4n+1}|\mathcal{R}_2|$ then use the comparison protocol described in section 2.4.3 to show $|\mathcal{R}_1| \gg |\mathcal{R}_2|$. Otherwise let $j = j + 1$. If $j \leq k$ tell V' to TRY NEXT ROUND, otherwise GIVE UP.

The set $\mathcal{R}_1$ can be thought of as all the possible random strings of V after round j of the protocol. The set $\mathcal{R}_2$ consists of the possible random strings of V generated by M. More formally:

Let $\mathcal{R}$ be the set of all possible coin tosses of V.
Let $\mathcal{R}_1 = \{R \in \mathcal{R} \mid R \text{ and } \alpha_1, \ldots, \alpha_{j-1} \text{ cause } V \text{ to say } \beta_1, \ldots, \beta_j\}$.
Let $\mathcal{R}_2 = \{R \in \mathcal{R} \mid M \text{ can output } R, \beta_1, \alpha_1, \ldots, \beta_j, \alpha_j \text{ part of a valid, accepting conversation}\}$

Note $\mathcal{R}_2 \subseteq \mathcal{R}_1$ and if $x \in L$ then $\mathcal{R}_2 \approx \mathcal{R}_1$. Also note $\mathcal{R}_1$ is independent of α_j.

We can test containment of an element in $\mathcal{R}_1$ in polynomial time and if $x \in L$ then M produces the exact distribution between P and V and thus r is a random element of $\mathcal{R}_2$ which P' does not know fulfilling the requirements of the comparison protocol. If $x \notin L$ possibly r is not a random element of $\mathcal{R}_2$ which can only increase the probability of the comparison protocol accepting.

2.5.4 Proof of the Protocol's Correctness

To show the protocol of section 2.5.3 forms an interactive protocol for $\overline{L}$, we must show:

1. If $x \in \overline{L}$ then $P' \leftrightarrow V'(x)$ accepts with probability $\geq \frac{2}{3}$
2. If $x \notin \overline{L}$ then for all $\hat{P}$, $\hat{P} \leftrightarrow V'(x)$ accepts with probability $\leq \frac{1}{3}$

1. Suppose to the contrary $x \notin L$ and the protocol fails to accept. If $|\mathcal{R}_1| < 2^{4n+1}|\mathcal{R}_2|$ then by the comparison lemma the comparison protocol will fail with an exponentially small probability. So $|\mathcal{R}_1| \geq 2^{4n+1}|\mathcal{R}_2|$ at all rounds j with probability at least one fourth. We will derive a contradiction by demonstrating $P \leftrightarrow V$ is not an interactive proof system for L by presenting a prover P^* which will convince V (the original verifier) $x \in L$ with probability greater than 2^{-6kn}.

At round j suppose the conversation so far has been $\beta'_1, \alpha'_1, \ldots, \beta'_j$. P^* works as follows:

P^*: Run M which outputs $r, \beta_1, \alpha_1, \ldots, \beta_k, \alpha_k$. Check this is a valid accepting conversation. If not, try again. See if $\beta_1, \alpha_1, \ldots, \beta_j = \beta'_1, \alpha'_1, \ldots, \beta'_j$. If not, try again.

$P^* \rightarrow V$: α_j

If $x \in L$ then the prover P^* will eventually succeed at each round because M must generate every possible conversation with some positive probability.

At round j when P^* has a conversation from M matching the conversation so far, $\mathcal{R}_1$ is the set of possible random coin tosses of V. When P^* says α_j, $\mathcal{R}_2$ is the set of coin tosses of V that will still keep V heading towards an accepting path. Since $|\mathcal{R}_2| \geq 2^{5n+1}|\mathcal{R}_1|$, this will happen with probability $\geq 2^{-(5n+1)}$. So after k rounds, V will end up accepting with a probability at least $\frac{1}{4}2^{-(5kn+k)}$ which is higher than the 2^{-6kn} maximum accepting probability we assumed for V and any P^* when $x \notin L$.

Note P^* may require exponential expected time to complete its part of the protocol but in our model we allow an infinitely powerful P^*.

2. Suppose $x \in L$. The simulator M will produce views of conversations from exactly the same distribution as P and V. Thus every conversation produced by M will be valid. Assume a prover $\hat{P}$ can convince V' to accept with probability $\geq \frac{1}{3}$. The verifier V will reject in this conversation with an exponentially small probability causing V' to accept. If $|\mathcal{R}_1| \leq 2^{n-4}|\mathcal{R}_2|$ on any round then the comparison protocol will accept also with an exponentially small probability. Thus we can assume with probability $> \frac{1}{4}$ that $|\mathcal{R}_1| > 2^{n-4}|\mathcal{R}_2|$ for some round j. Since M outputs all possible conversations, $\mathcal{R}_2$ is just the random coin tosses of V which might cause V to accept in the future. So at round j of the protocol, V accepts with probability less than $\frac{|\mathcal{R}_2|}{|\mathcal{R}_1|} \leq 2^{4-n}$. Since this happens at least a fourth of the time in general V accepts with probability at most $\frac{3}{4} + 2^{4-n}$ contradicting the fact V will accept with probability greater than $1 - 2^{-6kn}$. □

2.6 Extensions and Corollaries

Theorem 2.6 *Suppose $P \leftrightarrow V$ is an interactive proof system for a language L and there is a probabilistic polynomial-time simulator M such that $M[x]$ is statistically close to $P \leftrightarrow V[x]$. Then there is a single-round interactive proof system for the complement of L.*

Idea of Proof This extends the main theorem in two ways. First, we do not require $M[x] = P \leftrightarrow V[x]$, just they be statistically close. One can check the proof in the previous section and notice, with some minor adjustments to the probabilities, statistically close is good enough.

Second, we would like to get a single-round proof system for the complement of L. Notice in the protocol in section 2.5.3 the number of rounds depends on when P' decides to say STOP. To get bounded rounds we must make the following change to the protocol:

V': Run M k^3 times independently and get k^3 views of conversations; check each conversation is valid and accepting.

$V' \rightarrow P'$: For each i, $1 \leq i \leq k^3$, send the first i mod k rounds of the ith conversation.

$P' \rightarrow V'$: Pick any conversation j and show $|\mathcal{R}_1| \gg |\mathcal{R}_2|$ for the view of this conversation.

The proof still works because the new protocol essentially tries all rounds in parallel. Once we have bounded rounds we apply the theorems of [B, GS] that imply single-round protocols can simulate any bounded-round protocol.

Some trivial corollaries that follow from results described in sections 1.5 and 2.3:

Corollary 2.7 *If L has an statistical zero-knowledge interactive proof system (possibly with an unbounded number of rounds) then*

1. *the complement of L has a one-round interactive proof system.*
2. *L is contained in the intersection of almost-NP and almost-co-NP.*

Corollary 2.8 *If any NP-complete language has an statistical zero-knowledge interactive proof system then the polynomial-time hierarchy collapses to the second level.*

Corollary 2.9 *If one-way functions exist and the polynomial-time hierarchy does not collapse then $NP \subseteq ZK$; but $NP \not\subseteq SZK$, so $ZK \neq SZK$.*

2.7 Further Research

There are several interesting problems remaining concerning perfect zero-knowledge, including:

- What is the relationship between PZK and SZK?
- Are complement of perfect or statistical zero-knowledge languages themselves perfect zero-knowledge in any sense?
- Do we need cryptographic assumptions to show NP has zero-knowledge proof systems? Although this chapter shows NP probably does not have perfect zero-knowledge proof systems; possibly we need only assume the intractability of SAT for a zero-knowledge proof system.

Chapter 3

Logarithmic-Space Verifiers

3.1 Reducing the Power of the Verifier

Often in complexity theory, as in real life, we would like our computers to require small work space as well as a short amount of time. In this chapter, we look at the complexity of verifiers not only restricted in polynomial time but logarithmic space.

Condon and Ladner [CL] first looked at these space bounded proof systems in 1986. In 1987, Condon [Co1, Co2] described properties of these models with different results for public and private coins in contrast to the Goldwasser and Sipser result [GS].

We will concentrate on one model with the verifier restricted to public coins, polynomial time and logarithmic space. We show the equivalence of this model to adding both nondeterministic and probabilistic computation to logarithmic space bounded Turing machines.

3.2 Log Space Verifiers and BPNL

We define an interactive protocol with a log-space verifier as an interactive protocol with the following complexity for the verifier V:

1. V uses at most $O(\log n)$ space.
2. V runs in polynomial time.
3. V uses only public coins.

The class IPL contains all the languages accepted by these models using the same probabilities as for the standard model described in section 1.3.7.

The last two restrictions on V prevent the model from becoming too powerful. If we allow V unrestricted time with public coins this model accepts exactly the class P [Co1]. Anne Condon and John Rompel independently have shown if we allow V to use private coins then this model accepts the same languages as standard interactive proof systems.

Analogous to BPNP, we can define a probabilistic nondeterministic version of logarithmic space. We define the class of languages accepted by bounded-error probabilistic nondeterministic polynomial-time log-space machines as BPNL and we can show the following relationship between this model and interactive proof systems with log-space verifiers.

Theorem 3.1 *The classes IPL and BPNL contain the same set of languages.*

Proof

1. Suppose a language L has a BPNL machine M. We create a verifier that simulates M and when M makes a nondeterministic choice we defer that choice to the prover. On any input x, the probability of acceptance of this protocol is the same as the probability of acceptance of M since playing optimally, the prover chooses to send the message that leads to the highest probability of acceptance for V.

2. Suppose a language L has a IPL proof system with verifier V. We create M that simulates V but uses nondeterminism to guess the prover's responses. Again the probabilities of acceptance are identical. □

For the rest of this chapter we will use BPNL to refer to this class of languages.

3.3 A Circuit Model for BPNL

In this section, we describe a circuit model that captures the power of BPNL. We look at polynomial size circuits consisting of *max* and *average* gates instead of *and* and *or* gates. A max or average gate can take inputs from any value between 0 and 1 and outputs some value between 0 and 1. On inputs x_1 and x_2 a max gate will output $\max(x_1, x_2)$. On inputs x_1 and x_2 an average gate will output $1/2(x_1 + x_2)$. *Max-ave* circuits are a family of polynomial size circuits of max and average gates $C_1, C_2, \ldots$ with the following property of acceptance for a language L:
If $x \in \{0,1\}^n$ and the bottom most inputs of C_n are just $x_1, \ldots, x_n$, the bits of x, then

1. If $x \in L$ then $C_n(x_1, \ldots, x_n) \geq \frac{2}{3}$

2. If $x \notin L$ then $C_n(x_1, \ldots, x_n) \leq \frac{1}{3}$

As with probabilistic computation we require that C_n never falls between $\frac{1}{3}$ and $\frac{2}{3}$ for any input. We call circuits with this property *proper* max-ave circuits.

One can make an analogy from max-ave circuits to BPNL by having the prover make the choices on the max gates and the verifier randomly making a choice in the average gate. We make this notion formal as follows:

Theorem 3.2

1. *For every L in BPNL there is a log-space computable function f such that $f(x) = C$ where C is a proper max-ave circuit for L with size polynomial in $|x|$ and no nonconstant inputs.*

2. *There is a BPNL machine M such that* $\Pr(M \text{ accepts } (C, x)) = C(x)$ *for all $x \in \{0,1\}^*$ and max-ave circuits C.*

Proof

1. We will show given a BPNL machine M and an input x, we can create in log-space a circuit C such that the probability of M accepting is the value of C. We add a clock to M to keep track of how many steps have gone by. Let S be the set of all possible configurations of M. Clearly there are only a polynomial number of possibilities for S. These configurations will form the gates of the circuit C. Those configurations where M nondeterministically guesses a bit form max gates over all the next possible configurations. The configurations where M flips a coin form average gates over the next possible configurations one step after the coin flip. We replace an accepting configuration by the constant input one. Likewise, we replace the rejecting configurations by zero. The other configurations form a max gate over the single gate representing the next configuration after a single step of deterministic computation. Since we have added a clock to M, this process can not create any cycles. By the definition of BPNL and max-ave circuits, the probability of M accepting is the value of C. We clearly can do the above construction in log-space. Note C does not have any nonconstant inputs.

2. We will show that we can create a prover-verifier pair to accept x with probability $C(x)$. The prover and verifier will start at the top gate. For a max gate the prover picks one of that gate's children. For an average gate, the verifier picks one of the gate's children by flipping a coin. We continue this process on the gate's child. Since C has only a polynomial number of gates, V can always keep a pointer to the gate currently being processed. The verifier will accept if we process a one input. Again the probability of acceptance is exactly the value $C(x)$. We then turn (P, V) into an equivalent BPNL machine M. □

Corollary 3.3 *BPNL $\subseteq$ P (Proven independently by Condon [Co2])*

Proof Given L in BPNL and x in Σ^*, by the previous theorem we have a reduction $f(x)$ to a circuit C with no nonconstant inputs. We then compute C by calculating the value at each gate and accept if the value of C is at least two thirds. □

Note this does not say max-ave circuits accept exactly the same languages as BPNL machines. For example, a max-ave circuit can only accept monotone languages, thus no family of max-ave circuits can exist accepting nonmonotone languages such as parity. However, BPNL can clearly compute parity. By the above proof a BPNL machine can accept any language accepted by *log-space uniform* proper max-ave circuits.

Does BPNL accept all the languages of P? We conjecture there exist polynomial-time computable languages not in BPNL. The following theorem indicates we will not easily settle this question by computing the value of the max-ave circuit.

Theorem 3.4 *$\{(C, x, v) | C(x) = v\}$ is log-space-complete for P.*

Proof Ladner [L] showed $\{(C', x) | C'(x) = 1\}$ log-space complete for P where C' is a normal polynomial size and-or circuit. We create C by replacing every *or* gate by a max gate and every *and* gate by an average gate. Then $C'(x) = 1$ if and only if $C(x) = 1$. □

3.4 BPNL Contains LOGCFL

Clearly BPNL contains both NL and BPL. In this section we show that BPNL nontrivially contains the complexity class LOGCFL, languages log-space reducible to context-free languages [Su1, Su2].

Venkateswaran [V] showed the equivalence of LOGCFL and the class of languages accepted by log-space uniform semi-unbounded log depth circuits, *i.e.* a family of and-or circuits of $O(\log n)$ depth and bounded fan-in *and*'s (possibly unbounded fan-in *or*'s). We can assume that the fan-in of the and's is two.

The class LOGCFL contains NL and equality is unknown. It is also unknown whether BPL contain LOGCFL.

Theorem 3.5 *$LOGCFL \subseteq BPNL$*

Proof Let L $\in$ LOGCFL. For x, let C be the appropriate log-space constructible semi-unbounded circuit such that $C(x) = 1$ iff $x \in L$. The prover and verifier will start at the top gate. If it is an *or* gate then the prover picks one of that gate's children. If it is an *and* gate the verifier picks one of the gates children by flipping a coin. We then continue this process on the gate's child. Since C is of polynomial size, V can always keep a pointer to the gate currently being processed. We will end up at either

1. An input x_i. In this case the verifier accepts if $x_i = 1$.
2. A negation of an input x_i. In this case the verifier accepts if $x_i = 0$.

If $x \in L$ then for any choices at *and* gates, there exist choices at *or* gates that cause the circuit to accept. Thus the prover has a winning strategy for any choices of the verifier so the verifier always accepts.

If $x \notin L$ then there are choices the verifier V could make so V would reject. Since the circuit is only log depth and V makes one choice out of two at each gate the probability that V would reject$\geq 2^{-O(\log n)} \geq \frac{1}{p(n)}$ for some polynomial $p(n)$. Thus Pr(P and V accept x) $\leq 1 - \frac{1}{p(n)}$.

We then run this protocol $2p(n)$ times independently and in succession and V accepts if all runs of this protocol accept. We then get:

If $x \in L$ then Pr(P and V accept x) $= 1 > \frac{2}{3}$

If $x \notin L$ then Pr(P and V accept) $\leq (1 - \frac{1}{p(n)})^{2p(n)} \approx e^{-2} < \frac{1}{3}$

P and V form a polynomial-time log-space public-coin interactive proof system for L. □

3.5 Further Directions of Research

Several open questions remain, including:

- We showed LOGCFL $\subseteq$ BPNL $\subseteq$ P. Can this gap be tightened?
- Are there other restrictions on the verifier that give rise to other interesting complexity classes?

- What is the relationship of max-ave circuits to other circuit models?
- It is clear that BPL is closed under complement and Immerman [I] recently showed the same was true for NL. Can similar techniques be used to show BPNL is closed under complement?

Chapter 4

Multiple Provers

4.1 Corroborating Suspects

Consider the case of two criminal suspects who are under interrogation to see if they are guilty of together robbing a bank. Of course they (the provers) are trying to convince Scotland Yard (the verifier) of their innocence. Assuming that they are in fact innocent, it is clear that their ability to convince the police of this is enhanced if they are questioned in separate rooms and can corroborate each other's stories without communicating.

Instead of the verifier communicating with only one prover, we will now look at the model where the verifier can communicate with many provers that can not communicate with each other. Ben-Or, Goldwasser, Kilian and Wigderson [BGKW] originally developed multi-prover interactive proof systems primarily for cryptographic purposes. They show every language accepted by a two prover interactive proof system has a perfect zero-knowledge two prover proof system (see section 2.2 for definitions of perfect zero-knowledge). They also show two prover systems can simulate any multi-prover system. Along the same lines of Goldreich, Mansour and Sipser [GMS], they show any two prover system has an equivalent system that accepts with probability one for strings in the language. Complete proofs of these results appear in [Ki].

We give a simple characterization of the power of the multi-prover model in terms of probabilistic oracle Turing machines. Using this characterization we give an oracle relative to which there exists a co-NP language not accepted by any multi-prover interactive proof system extending the result of Fortnow and Sipser [FS1] for the standard interactive proof system model.

4.2 Definitions

Let $P_1, P_2, \ldots P_k$ be infinitely powerful machines and V be a probabilistic polynomial-time machine, all of which share the same read-only input tape. The verifier V shares communication tapes with each P_i, but different provers P_i and P_j have no tapes they can both access besides the input tape. We allow k to be as large as a polynomial in the size of the input; any larger and V could not access all the provers.

Formally, similar to the prover of a standard interactive proof system, each P_i is a function

from the input and the conversation it has seen so far to a message. We put no restrictions on the complexity of this function other than that the lengths of the messages produced by this function must be bounded by a polynomial in the size of the input.

$P_1, \ldots, P_k$ and V form a multi-prover interactive protocol for a language L if:

1. If $x \in L$ then $\Pr(P_1, \ldots, P_k \text{ and } V \text{ on } x \text{ accept}) > 1 - 2^{-n}$.
2. If $x \notin L$ then for all provers $P'_1, \ldots, P'_k$, $\Pr(P'_1, \ldots, P'_k \text{ and } V \text{ on } x \text{ accept}) < 2^{-n}$

MIP is the class of all languages which have multi-prover interactive protocols. If k is one we get the class IP of languages accepted by standard interactive proof systems.

Note the different probabilities used here compared to the probabilities used to define standard interactive proof systems. Unlike the result of Babai and Moran [BM] for the standard model, it is unknown whether we can increase the probability of error in multi-prover proof systems by running the protocols in parallel (see section 4.5). We can reduce the probability of error to less than $2^{-p(n)}$ for any polynomial $p(n)$ by running the protocols several times serially.

A *round* of an multi-prover interactive protocol is a message from the verifier to some or all of the provers followed by messages from these provers to the verifier. In general, interactive protocols can have a polynomial number of rounds. We let α_{ij} designate a message from prover i to the verifier in round j and β_{ij} designate a message from the verifier to prover i in round j. We may omit the prover number for one-prover interactive protocols.

IP(j, k) is the class of languages accepted with no more than j provers in no more than k rounds. The values j and k may depend on the input but clearly can not be larger than a polynomial in the size of the input. We let *poly* designate a polynomial, *i.e.* IP is IP(1,*poly*) and MIP is IP(*poly*,*poly*). A protocol is said to have bounded rounds if k is a constant.

4.3 Probabilistic Oracle Machines

Suppose a prover in an interactive proof system must set all his possible responses before the protocol with the verifier takes place. We can think of the prover as an oracle attempting to convince a probabilistic machine whether to accept a certain input string. The oracle must be fully specified before the protocol begins.

Let M be a probabilistic polynomial-time Turing machine with access to an oracle O. A language L is accepted by an oracle machine M iff

1. For every $x \in L$ there is an oracle O such that M^O accepts x with probability $> 1 - 2^{-n}$
2. For every $x \notin L$ and for all oracles O', $\mathrm{M}^{O'}$ accepts with probability $< 2^{-n}$

This model differs from the standard interactive protocol model in that the oracle must be set ahead of time while in an interactive protocol the prover may let his future answers depend on previous ones.

Theorem 4.1 *L is accepted by an oracle machine if and only if L is accepted by a multi-prover interactive protocol.*

Proof (Write-up due to John Rompel)

($\Leftarrow$)

Suppose L is accepted by a multi-prover interactive proof system V. Then define M as follows: M simulates V with M remembering all messages. When V sends a message to a prover, M asks the oracle the question $(x, i, j, \ell, \beta_{i1}, \ldots, \beta_{ij})$ suitably encoded and uses the response as the ℓ^{th} bit of the j^{th} message from prover i on input x where $\beta_{i1}, \ldots, \beta_{ij}$ are the first j messages sent from the verifier to prover i. M then accepts x if and only if V does.

1. Let $P_1, \ldots, P_k$ be provers which cause V to accept each $x \in L$ with probability at least $1 - 2^{-n}$. If we let O be the oracle which encodes in the above manner the messages of $P_1, \ldots, P_k$, then M^O will accept each $x \in L$ with the same probability as V.

2. Suppose there were an input $x \notin L$ and an oracle O' such that $M^{O'}$ accepts x with probability more than 2^{-n}. Then we could construct provers $P'_1, \ldots, P'_k$ which cause V to accept x with the same probability by just using O' to create their messages. Since, by definition, no such $P'_1, \ldots, P'_k$ exist, neither does O'.

($\Longrightarrow$)

Suppose L is accepted by a probabilistic oracle machine M in n^k steps. We will define a verifier, V, to simulate M using n^{k+1} provers. The verifier first randomly chooses an ordering of the n^{k+1} provers. The verifier then simulates M and whenever M asks an oracle question, V asks the question to each of the next n provers in the chosen ordering. If the provers are unanimous in their answer, V uses that answer in its simulation of M; if not, V rejects immediately. If the provers successfully answer all oracle queries, then the verifier accepts if and only if M does. There can be at most n^k questions so the n^{k+1} provers will suffice.

1. Let O be an oracle such that M^O accepts each $x \in L$ with probability at least $1 - 2^{-n}$. If we let $P_1, \ldots, P_{n^{k+1}}$ all answer (identically) according to O, then they will cause V to accept each $x \in L$ with the same probability as M^O.

2. Consider $x \notin L$; consider any provers $P'_1, \ldots, P'_{n^{k+1}}$. Let oracle O' answer queries as the majority of $P'_1, \ldots, P'_{n^{k+1}}$ would.

   There are two cases to consider for V accepting x: either all oracle queries in the simulation answered consistent with O' or some oracle query answered different than O' would. By the definition of acceptance for probabilistic oracle machines, we know that the probability of the first case occurring is 2^{-n}, where this probability is over the random coins of M.

   Now consider the second case. Fix some set of random coins r. Let $q_1, \ldots, q_{n^k}$ be the oracle questions in the computation of M on x using random coins r and oracle O'. For V to accept using an oracle answer inconsistent with O', it must be the case that, for some i, the i^{th} set of n provers all give an answer inconsistent with O' on q_i. Fix i. Since O' gives the answer to q_i that the majority of provers do, at most $\frac{1}{2}n^{k+1}$ of the n^{k+1} provers will answer differently than O'. Thus the probability that the i^{th} set of n provers will all answer differently from O' is at most

$$\frac{\binom{n^{k+1}/2}{n}}{\binom{n^{k+1}}{n}} \leq 2^{-n}$$

The probability that this will happen for some i is at most n^k times this. Thus the total probability that V will accept x is at most $2^{-n} + n^k 2^{-n}$ or $(n^k + 1)2^{-n}$.

Finally, we define V' to simulate V three times in series, accepting or rejecting according to the majority of the simulations. Since the probabilities of the runs are independent, the probability that the correct provers will cause V' to accept $x \in L$ is at least

$$(1-2^{-n})^3 + 3(1-2^{-n})^2 2^{-n} > 1 - 3 \cdot 2^{-2n}$$

and the probability that any provers will cause V' to accept $x \notin L$ is at most

$$2^{-3n} + 3 \cdot 2^{-2n}(1-2^{-n}) < 3 \cdot 2^{-2n}$$

If we further modify V', hardwiring the correct answer for $n \leq 1$, then V' forms a multi-prover interactive proof system for L. □

This theorem gives a natural model equivalent to multiple provers and useful for proving theorems about them.

4.4 Are there Multi-Prover Protocols for co-NP Languages?

With the extra power of multiple provers, we can not immediately rule out the possibility of protocols for all co-NP languages. However, we can extend the result of Fortnow and Sipser [FS1] where an oracle is given such that co-NP$^A \not\subseteq$ IP^A.

Theorem 4.2 *There exists an oracle A and a language $L \in$ co-NP^A such that $L \notin$ MIP^A.*

Proof In this proof we will use the oracle machine model. It is easy to verify that the proof in section 4.3 holds under relativizations to all oracles. Note that our machines can ask questions about two oracles, the "prover" oracle O and the "relativization" oracle A.

We can enumerate all possible polynomial-time machines in the standard manner, letting M_i be bounded in time by n^i, where n is the size of the input.

For any oracle A, let

$$L(A) = \{1^n : \text{A contains all strings of length } n\}$$

It is clear that $L(A) \in$ co-NP^A for all oracles A.

In step i we make $L(A)$ different from every oracle machine M_i^A. Then $L(A)$ can not have a multi-prover interactive protocol and we have proved our theorem.

STEP i:

Pick N_i large enough so $2^{N_i} > 3(N_i)^i$ and no oracle questions of length N_i have been asked in any previous step. Let $p_i = (N_i)^i$.

Every time M_i^A asks a question to A which has not been previously answered we answer yes. If there are not any oracles O such that O and M_i^A accept on input 1^{N_i} with probability at least $\frac{2}{3}$ then we put in the oracle A all strings of length N_i and every other previously unset string that M_i^A asks about for any oracle O. This completes step i. Note that M_i^A can only ask questions of length less than p_i so we will always be able to find N_{i+1} in step $i+1$.

Otherwise we have some oracle O such that O and M_i^A will accept 1^{N_i} with probability at least $\frac{2}{3}$. On any computation path (which is determined by M_i^A's coin tosses), M_i^A can ask at most p_i oracle questions to A of length N_i. There are 2^{N_i} questions of length N_i. A counting argument shows that there is some oracle question x of length N_i that appears in no more than $p_i/2^{N_i}$ of the computation paths of M_i^A. By the way we chose N_i this means the oracle question x appears in less than one third of the computation paths of M_i^A. Put all strings of length N_i except for x in the oracle A. Also place in the oracle A every string queried by M_i^A on every possible communication with every possible O. The oracle O will convince M_i^A to accept with probability greater than one third since more than a third of the computation paths are the same as before.

If there exists an oracle O that makes M_i^A accept 1^{N_i} with probability greater than two thirds then $L(A)$ does not contain 1^{N_i}. Conversely, if no oracles exists that causes M_i^A to accept with probability at least one third then $L(A)$ will contain 1^{N_i}. By the standard diagonalization argument $L(A)$ does not equal the language accepted by M_j^A for any j. □

This result implies the earlier result of Fortnow and Sipser [FS1] since the language $L(A)$ does not have standard interactive proof systems under the oracle A.

Corollary 4.3 *Techniques which relativize will not settle whether MIP contains co-NP or whether IP contains co-NP.*

Proof Let B be the standard oracle which makes $\mathrm{P}^B = \mathrm{NP}^B$ [BGS]. Then co-$\mathrm{NP}^B = \mathrm{P}^B$ and co-$\mathrm{NP}^B \subseteq \mathrm{IP}^B \subseteq \mathrm{MIP}^B$. Thus any proof that proves or disproves co-NP $\subseteq$ MIP or co-NP $\subseteq$ IP can not relativize. □

4.5 Bounded Round Protocols

Fortnow, Rompel and Sipser [FRS] claimed some results about collapsing rounds: IP(1,*poly*) $\subseteq$ IP(2,1) and IP(*poly*,*poly*) = IP(3,2). The "proofs" require that we can somehow decrease the error probability by running the protocols in parallel. The approach makes the assumption that if the provers can be prevented from communicating among themselves through the protocol then parallel runs of the protocol work independently like parallel runs of one prover interactive protocols [BM].

The claims of Fortnow, Rompel and Sipser remain unproven because of this faulty assumption. We show the assumption faulty with the following counterexample:

Suppose we have the following two prover protocol:

V: Pick two bits a and b uniformly and independently at random.

$V \rightarrow P_1$: a

$V \rightarrow P_2$: b

$P_1 \rightarrow V$: c

$P_2 \rightarrow V$: d

V: Accept if $(a \vee c) \neq (b \vee d)$.

It is easy to show the best strategy for two provers causes the verifier to accept with probability 1/2. Notice neither prover has any notion of what bit the verifier has sent to the other prover.

Now let us examine the two round version of the same protocol:

V: Pick bits a_1, a_2 and b_1, b_2 uniformly and independently at random.

$V \rightarrow P_1$: a_1, a_2

$V \rightarrow P_2$: b_1, b_2

$P_1 \rightarrow V$: c_1, c_2

$P_2 \rightarrow V$: d_1, d_2

V: Accept if $(a_1 \vee c_1) \neq (b_1 \vee d_1)$ and $(a_2 \vee c_2) \neq (b_2 \vee d_2)$.

If the parallel runs of the protocol behave independently we would expect the optimum strategy for the provers causes the verifier to accept with probability $(1/2)^2 = 1/4$. However the following strategy for the provers causes the verifier to accept with probability 3/8:

P_1: If $a_1 = a_2 = 0$ respond $c_1 = c_2 = 0$ otherwise respond $c_1 = c_2 = 1$.

P_2: If $b_1 = b_2 = 0$ respond $d_1 = d_2 = 0$ otherwise respond $d_1 = d_2 = 1$.

Note in n rounds the probability of acceptance of this protocol can not exceed $(3/4)^n$ since the verifier will not accept if $a_i = b_i = 1$ for any i. We can not find any counterexample without this type of exponential decrease. However we can not prove any such decrease in a general setting.

We conjecture the bounded round claims of Fortnow, Rompel and Sipser are true but the proofs will require new techniques.

4.6 Further Research

There still remain many open questions including:

- Can we in fact prove the results like those stated in [FRS], *i.e.* results that collapse unbounded rounds to bounded rounds perhaps with additional provers?
- Under what conditions can we reduce the error probability without drastically increasing the number of rounds in multi-prover interactive proof systems?
- What is the relation between MIP and IP? Is there, for instance, an oracle separating the two classes?
- What is the relationship between MIP and PSPACE? Feldman [Fe] shows that PSPACE contains IP, but the proof does not appear to work for MIP. Peterson and Reif [PR] show if we replace the verifier's randomness with universal choices we get exactly non-deterministic exponential time.
- A public-coin interactive proof system can accept any language accepted by a interactive proof system [GS]. What can we say about public-coin multi-prover interactive proof systems? How do we even define public-coin proof systems for multiple provers?

Chapter 5

Probabilistic Computation and Linear Time

5.1 Linear-Time Verifiers

Suppose we restrict our verifier to run in linear time. Clearly any language accepted by a verifier running in linear time will be accepted by a standard interactive proof system with a polynomial-time verifier. Can we find a language accepted by a standard interactive proof system but not a proof system with a linear-time verifier?

In this chapter we examine the same question for simpler models of computation. In 1965 in the seminal paper in complexity theory, Hartmanis and Stearns [HaS] showed for any $k \geq 1$ there exist problems with deterministic n^{k+1} algorithms but no deterministic algorithms exist that run in n^k steps. This hierarchy theorem answered the question for the simple deterministic case.

Interactive proof systems combine nondeterministic and probabilistic computation. In 1973, Cook [C2] showed a hierarchy exists for nondeterministic computation. In contrast with deterministic and nondeterministic computation, the existence of a probabilistic hierarchy remains unknown. The techniques that establish the deterministic and nondeterministic hierarchy fail in the probabilistic case. The main result of this chapter shows a fundamental reason for this failure.

We will show this result by exhibiting an oracle A relative to which probabilistic linear time equals BPP, probabilistic polynomial time, as well as an oracle for which they differ. Thus techniques that relativize will not answer this question. Virtually all known techniques for solving problems of this type relativize, particularly the techniques that separate the deterministic and nondeterministic time classes.

This result suggests the possibility of a collapse of a complexity time hierarchy. Results of this nature show some fundamental differences in probabilistic computation versus other forms of computation such as deterministic and nondeterministic.

Assuming computers have easy access to random bits, a problem has an efficient solution if a probabilistic polynomial-time algorithm can solve this problem. Under the oracle A, we have the surprising situation that we can solve all problems with efficient solutions in probabilistic linear time.

We also show some other relativized results in this paper relating to probabilistic computation and linear time. We also give a partial answer to the original question by showing the existence of a language accepted by an interactive proof system but not accepted in probabilistic linear time.

5.2 Our Results and Related Results

We show the existence of oracles under which the following hold:

1. BPP = BPTIME[n] (actually we will show BPP = RTIME[n] (thus BPP $\subseteq$ NTIME[n]) and BPP = ZPTIME[n])
2. BPP has linear size circuits.
3. $\Delta_2 \subseteq$ BPTIME[n]
4. The negation of each of the above.

We also show there must exist a language in either BPP or NP but not in BPTIME[n]. This result implies there are languages accepted by interactive proof systems that are not accepted in probabilistic linear time.

Hartmanis and Stearns [HaS] with Hennie and Stearns [HeS] show for "nice" f and g such that $g(n) = o(\frac{f(n)}{\log n})$ that DTIME[$f(n)$] $\not\subseteq$ DTIME[$g(n)$], thus DTIME[n^j] $\not\subseteq$ DTIME[n^k] for $1 \leq k < j$. Cook [C2] showed the latter result for nondeterministic time, which was improved by Seiferas, Fischer and Meyer [SFM]. All of these results relativize to all oracles.

Wilson [W] showed Δ_2 has linear size circuits with an appropriate oracle. We extend Wilson's result to show BPTIME[n] contains Δ_2 relative to an oracle. However, his techniques fail to help prove our main theorem since they rely on the fact that languages in Δ_2 can depend only on a polynomial number of oracle queries for each input. BPP does not afford us that luxury.

Kannan [K] showed $\Sigma_2 \cap \Pi_2$ does not have n^k–size circuits for any fixed k. Using standard techniques, one can show BPTIME[n] has n^4–size circuits. These results relativize to all oracles. Combining these facts with the above results, we get that there are oracles that put Δ_2 and BPP in BPTIME[n] and linear size circuits where as such oracles do not exist for $\Sigma_2 \cap \Pi_2$. The class $\Sigma_2 \cap \Pi_2$ contains BPP [Si] and Δ_2 though the relationship between Δ_2 and BPP is unknown.

One can get a trivial separation of BPTIME[n] and BPTIME[2^n] by simulating all possible coin tosses. Karpinski and Verbeek [KV] improved this result to show BPTIME[$n^{\log n}$] does not contain BPTIME[2^{n^ϵ}] for any $\epsilon > 0$.

5.3 Deterministic, Nondeterministic and Probabilistic Linear Time

Why does probabilistic computation behave differently than deterministic and nondeterministic computation for separating the time classes? In this section we will describe the proof

techniques for separating the deterministic and nondeterministic time classes and show why these techniques fail for probabilistic computation.

The proof that DTIME[n^2] $\not\subseteq$ DTIME[n] works roughly as follows: Let $M_1, M_2, \ldots$ be an enumeration of all linear-time deterministic Turing machines. Define a machine M that on input i does the following: Simulate M_i on input i and accept if and only if M_i rejects. In quadratic time, M has more than enough time to simulate M_i on input i. However, if a linear-time machine M_j accepts $L(M)$ then we have a contradiction by the definition of M.

At first glance this proof seems to work for probabilistic machines. However, the proof fails because of the enumeration of the machines. If we choose a standard enumeration of the BPP machines, one of the M_i will accept with probability 1/2 (for example the machine that just flips a coin and accepts if heads). Since a proper probabilistic machine must accept with probability below 1/3 or above 2/3, M will not be in BPTIME[n^2] even though it runs in quadratic time. We could try to have an enumeration of proper probabilistic linear-time machines but such an enumeration may be computationally infeasible.

The deterministic proof does not work for the nondeterministic case either. Cook [C2] proved NTIME[n^2] $\not\subseteq$ NTIME[n] using a translation lemma: If NTIME[$f(n)$] $\subseteq$ NTIME[$g(n)$] then for all "reasonable" superlinear h, NTIME[$h(f(n))$] $\subseteq$ NTIME[$h(g(n))$]. A similar lemma holds for deterministic and probabilistic computation. Cook's proof proceeds as follows: Assume NTIME[n^2] = NTIME[n]. Then by using the translation lemma NTIME[n^4] = NTIME[n^2] and thus NTIME[n^4] = NTIME[n]. If we repeat this process k times, we get NTIME[n^{2^k}] = NTIME[n]. By using an universal nondeterministic machine, we are able to maintain the same constant at each step, *i.e.* a nondeterministic machine that runs in n^{2^k} can be simulated by a machine that runs in $c^k n$ time for some fixed c. If we let $k = \log n$, we get NTIME[n^n] = NTIME[$n^{1+\log c}$] which can be shown false by diagonalization.

Even though the translation lemma holds, this proof still fails for probabilistic computation. The difficulty comes when we try to make a universal proper probabilistic machine. A universal proper probabilistic machine would be a proper probabilistic machine that can simulate other proper probabilistic machines; a very difficult task as we have already seen. Thus we can not keep the constants in check, and therefore can only repeat the translation process a constant number of times.

We will exploit these difficulties to create the oracle to collapse BPP to BPTIME[n].

5.4 Proof of the Main Theorem

In order to construct an oracle A such that BPP^A = BPTIME^A[n] we encode within A the answers to whether BPP^A machines M accepts inputs w (for each M, and almost all w) in a way that a BPTIME^A[n] machine can find the encoding and thus perform the simulation quickly. The difficulty that arises is that the BPP^A machine also has access to A and so can use it to try to ensure that however we try to encode the answers the simulating machine will be incorrect. This is in fact why analogous oracles for P and NP can not exist since the theorems of [HaS, C2] relativize for all oracles. Our ability to construct the oracle in this case rests on a balancing act between the power of probabilistic over deterministic computation on one side, its still limited ability on the second side and the "forbidden" region of acceptance probability (between 1/3 and 2/3) lastly.

We present the proof in several sections as follows:

1. We describe the structure of the oracle.
2. We examine a simple case in which machines and inputs only look at their own encodings.
3. We define influencing strings, oracle strings that affect the acceptance by a reasonable probability, and show proper machines can not depend on noninfluencing strings.
4. We describe the encoding process for a restrictive BPP machine.
5. We create a dependency graph for a machine and an input.
6. We process the dependency graph encoding that machine and input.
7. We generalize the proof to all BPP machines.

5.4.1 Structure of the Oracle

Let $M_1, M_2, \ldots$ be an enumeration of polynomial-time Turing machines that can make random choices and ask queries of an oracle. M_i^A designates the machine with index i using oracle A. Without loss of generality we can assume $M_i^A(x)$ runs in at most n^i steps. Clearly if $L \in \mathrm{BPP}^A$ then $L = L(M_i^A)$ for some i with M_i^A being a proper probabilistic Turing machine.

Without loss of generality we can assume $M_i^A(x)$ flips all its coins before it does any other computation. Once $M_i^A(x)$ has flipped these coins it becomes a deterministic machine whose acceptance depends solely on the oracle questions it asks. We call the computation after $M_i^A(x)$ flips its coins a *computation path* of $M_i^A(x)$. We also assume $M_i^A(x)$ flips the same number of coins on each computation path, so that each path has the same probability of occurring. Since $M_i^A(x)$ runs in polynomial time it can ask only a polynomial number of oracle queries on any computation path.

We will create the oracle A such that for each machine M_i^A one of the two following statements will be true:

1. M_i^A is improper.
2. There will exist a probabilistic linear-time machine S_i^A that accepts exactly the same language as M_i^A.

Suppose BPP^A contains a language L. Then some machine M_j^A must accept exactly the language L. The machine M_j^A must be proper, otherwise it could not accept any language at all, certainly not L. If we have set up the oracle A as described above then we have a linear-time machine S_j^A accepting the same language as M_j^A, *i.e.* the language L. Thus all of BPP collapses to probabilistic linear time under the oracle A.

We could encode whether $M_i^A(x)$ accepts by putting the string (i, x) in the oracle A if and only if $M_i^A(x)$ accepts. The linear-time machine S_i^A accepts on input x if (i, x) is in A. This idea fails to work because $M_i^A(x)$ can look at its own encoding of (i, x) in A.

Instead, we will encode in the oracle A whether or not M_i^A accepts using strings of the form (i, x, r) where r may be any of the $2^{5|x|}$ strings of length $5|x|$.

We say A *properly encodes* $M_i^A(x)$ if

1. If M_i^A accepts x then for at least 2/3 of the possible r's, $(i,x,r) \in A$.

2. If M_i^A rejects x then for at most 1/3 of the possible r's, $(i,x,r) \in A$.

We call the (i,x,r)'s *encoding* strings of $M_i^A(x)$.

We now have the simulating machine S_i^A do the following for input x:

1. Pick a random r of length $5|x|$.

2. Accept if (i,x,r) is in the oracle A.

Notice that if we have properly encoded A for M_i^A then S_i^A will accept exactly the same language as M_i^A. We will properly encode the oracle A for all proper polynomial-time probabilistic machines M_i^A.

5.4.2 A Simple Case

Let us examine the case when $M_i^A(x)$ only looks at strings of its own encoding, *i.e.* strings of the form (i,x,r).

An *influencing* string of $M_i^A(x)$ is an oracle query that occurs on at least one sixth of all the computation paths. An easy counting argument shows $M_i^A(x)$ can have at most a polynomial number of influencing strings and thus they make up a very small fraction of all the possible (i,x,r).

We will properly encode $M_i^A(x)$ in A as follows:

Initially we will set all of the strings of the form (i,x,r) to zero, *i.e.* not in A. We will then determine the probability of $M_i^A(x)$ accepting. There are three cases:

1. $M_i^A(x)$ accepts with probability less than one third.

2. $M_i^A(x)$ accepts with probability between one third and two thirds.

3. $M_i^A(x)$ accepts with probability greater than two thirds.

In the first case $M_i^A(x)$ rejects by definition and we have set less than a third of the (i,x,r) in A; in fact, none of the (i,x,r) are in A. In this case we have already properly encoded A.

In the second case $M_i^A(x)$, being improper, can not accept any languages, so we no longer need to encode A for M_i^A, even for other inputs.

In the third case, $M_i^A(x)$ accepts, but there are less than two thirds of the (i,x,r) in A. We will properly encode A using the following algorithm:

Let S be a collection of two thirds of the (i,x,r) such that S has only noninfluencing strings. The set S exists because the influencing (i,x,r) form only a tiny fraction of all of the (i,x,r).

Pick a single string from S and put that string in the oracle A. Now only one string of the form (i,x,r) is in the oracle. Once again determine the probability of $M_i^A(x)$ accepting.

If $M_i^A(x)$ accepts with probability between one third and two thirds, then M_i^A is improper and we no longer need to encode M_i^A.

$M_i^A(x)$ can not accept with probability less than one third. Since we chose an noninfluencing string occurring on at most one sixth of the computation paths of $M_i^A(x)$ it can not

change its probability by more than one sixth. However $M_i^A(x)$ would have to change its probability by more than one third to accept with probability less than one third.

Thus if $M_i^A(x)$ is still proper then it must accept with probability at least two thirds. We then pick another string from S, and put that string in the oracle A along with the first string.

Once again either $M_i^A(x)$ is improper or it accepts with probability more than two thirds. We continue this process until M_i^A becomes improper or we have added to A all of S. At this point $M_i^A(x)$ accepts and two thirds of the (i, x, r) are in A. Thus we have properly encoded A for $M_i^A(x)$.

Unfortunately, this does not finish the proof because $M_i^A(x)$ may ask questions of other machines and inputs. To handle this case we must look carefully at the dependencies among the machines and the encoding strings they query. The remainder of the proof handles these dependencies.

5.4.3 Influencing and Simulating Strings

For each $M_i^A(x)$ we will look at all the oracle queries it can possibly ask on every computation path—potentially a very large set of oracle strings. We call the oracle queries that affect the probability of $M_i^A(x)$ accepting by a certain nonnegligible probability *influencing strings*, a slight change from the definition in section 5.4.2. The remaining oracle queries we call *simulating strings.* We will show that $M_i^A(x)$ can not have too many influencing strings.

Suppose $M_i^A(x)$ depends on its simulating strings, *i.e.* one setting of these strings causes $M_i^A(x)$ to accept with probability less than 1/3 and another setting causes $M_i^A(x)$ to accept with probability more than 2/3. We can start with the first setting and change one oracle query at a time to get to the second setting. Eventually $M_i^A(x)$ must accept with probability greater than 1/3. However since the simulating strings do not individually change the probability of $M_i^A(x)$ very much, the probability $M_i^A(x)$ accepts can not be greater than 2/3 so M_i^A is no longer a proper probabilistic machine.

We now give an outline of the rest of the proof using influencing and simulating strings: For every machine M_i and input x, we try all possible settings of the oracle queries of A in an effort to make $M_i^A(x)$ accept with an improper probability. If we succeeded in making M_i^A improper we set all other encoding strings of M_i to zero. Otherwise M_i depends only on its influencing strings, we just set these arbitrarily in A to determine $M_i^A(x)$. We have not set too many strings; in particular we have not set very many of the (i, x, r)'s. We then use the unset (i, x, r)'s to encode $M_i^A(x)$ consistent with whether it accepts.

5.4.4 Order of Encoding

For any given proper M_i^A, we need only properly encode oracle A for all but a finite number of inputs for M_i^A. The linear-time probabilistic machine S_i^A that merely chooses a random r and accepts if the oracle A contains (i, x, r), will work for all the inputs properly encoded by A. We create a linear-time machine T_i^A that accepts the same language as M_i^A by "hardwiring" the answers of the finite number of inputs not properly encoded by A.

We will use a *finite injury* argument. For a given M_i^A, we might not properly encode M_i^A on some finite number of inputs. We will determine which inputs we will not properly encode as the construction happens, making sure only that a finite number of inputs are not

properly encoded. For example, suppose we can set the encoding strings of M_i^A, strings of the form (i, x, r), in order to make M_j^A improper for $i > j$. Then we do not have to consider M_j^A again. Since there is only $i - 1$ machines with a lower index than i, we can only set the encoding strings in this way for a finite number of times.

We will encode machines and inputs in order of input size. For inputs of length n, we will encode machines $M_1, M_2, \ldots, M_{\log\log\log n}$. Thus for machine i, we will not encode any of the finite number of inputs of size smaller than $2^{2^{2^i}}$.

For each $M_i^A(x)$ we will either

1. Set enough of the oracle A to determine whether $M_i^A(x)$ accepts and appropriately set the encoding strings of $M_i^A(x)$ in A.
2. Make M_i^A improper.
3. Make some M_j^A improper for some $j < i$.

At all times we carefully set the oracle questions in A as to not use too many encoding questions for any machine and/or input except for finite injury in cases 2 and 3.

For each machine and input we will look at its influencing and simulating strings. If a machine depends on simulating strings of a higher indexed machine by the earlier argument we can set these strings to make the machine improper; since we only changed strings of a higher index we use the finite injury argument. If a machine has simulating strings of a lower index we recursively encode those machines—note we limit ourselves to $\log\log\log n$ indices.

Ideally we would like to just set all the influencing strings in A immediately. Unfortunately, influencing strings of different inputs may take up a large part of the encoding strings of some $M_i(x)$. This may happen during the recursion of simulating strings. We show we only have to worry about influencing strings of encodings of smaller inputs; the others we will immediately set. We recursively encode those machines, showing the recursive depth can not be too deep to prevent the encoding of the original machine. Note we never set the simulating strings of a machine unless either we can make that machine improper or we encode a machine whose encoding strings are the simulating strings of another machine.

For now we will make the following restriction on how M_i works. We assume M_i does not make oracle queries dependent on the answers of previous queries. In other words, for any given computation path, M_i has a fixed set of oracle queries to decide whether to accept or reject. This set can have at most $|x|^i$ oracle queries, the running time of M_i on x. We will show later how to extend this proof to the general case where M_i's oracle queries can depend on previous queries.

We encode $M_i^A(x)$ in two phases. First we will determine which encoding strings $M_i(x)$ depends on. Then we properly encode these machines until we have encoded $M_i^A(x)$. We do this through use of a dependency graph.

Before we encode any machines we set all oracle strings not used for encoding to zero. Since we encode in order of input size, we have previously determined all of the following oracle strings:

1. Nonencoding strings of the oracle.
2. Encoding strings of the form (j, y, r) for all j, y and r such that $j > \log\log\log|y|$ since we only encode the first $\log\log\log n$ machines for inputs of length n.

3. Encoding strings of all inputs of size less than $n = |x|$ since these string have been previously encoded.

4. Encoding strings of all machines previously made improper.

As we will see in this proof, we may have determined strings in A in addition to those listed above.

5.4.5 Creating the Dependency Graph

We will create a finite *dependency graph* to help us properly encode $M_i^A(x)$. The nodes of the dependency graph represent a machine and an input. Directed edges go from one machine and input to another if the first machine on that input asks oracle queries of encoding strings of the other machine and input. Nodes will be of the form (j, y) representing machine M_j on input y. We call j the *index* of node (j, y) and y the *input*. We define an ordering of the nodes by $(j, y) < (k, z)$ if $j < k$ or $j = k$ and $y < z$ for some ordering of input strings such that $|y| < |z|$ implies $y < z$.

In the simple case we assumed the dependency graph had only self loops, *i.e.* machines and inputs only look at their own encoding strings. If the dependency graph had no cycles, we could properly encode all the nodes by encoding the leaves and then work our way back to the root. However, the dependency graph may have cycles, in which case we will require more work to process this graph.

To create the dependency graph G for $M_i^A(x)$ we first start with the single node (i, x). Let us place the graph on a two dimensional grid with indices increasing from left to right and inputs increasing from bottom to top. We place (i, x) in the lower right corner. We *expand* a node (j, y) by adding an edge to (k, z) (creating the node if necessary) if for some selection of random coins $M_j(y)$ asks an as yet undetermined encoding string of $M_k(z)$. We recursively expand node (k, z) if one of the following holds:

1. $k < j$, *i.e.* node (k, z) is to the left of node (j, y)

2. $k < i$ and $|z| < |y|$ and $|z| < |w|$ for some node (l, w) of G with $l \geq k$, *i.e.* node (k, z) is to the left of the original node (i, x), is below (j, y) and is below some node at least as far right as (k, z). We need this condition to prevent the indices from increasing without bound.

We get the dependency graph G by expanding the single node (i, x). We will see this graph contains the structure of the recursive encodings necessary to encode $M_i(x)$. A node (j, y) *depends* on node (h, z) if there is an edge from (j, y) to (h, z). Likewise, $M_j(y)$ *depends* on $M_h(z)$ if (j, y) depends on (h, z).

Note that all nodes of G represent machines with as yet undetermined encoding strings. If (j, y) is a node of G then $1 \leq j \leq \log\log\log n$ and $|y| \geq n$. If (j, y) is an expanded node of G other than (i, x) then $j < i$.

Lemma 5.1 *If* $(j, y) \in G$ *then* $|y| < n^{\log n}$.

Proof A Turing machine can not ask an oracle question longer than the amount of time it has to write it down. If (k, z) is a node of G then M has running time at most $|z|^{\log\log\log|z|}$

since $k \leq \log\log\log n$ and M_k runs in time at most n^k for inputs of size n. By the definition of G, we only expand a node with a larger input if the index is smaller. If $f(j)$ is the maximum size of the inputs of all the nodes of index at least j then $f(j-1) \leq f(j)^{\log\log\log f(j)}$. We know $f(i) = n$ since (i, x) is the only expanded node of G of index i. We can bound the recurrence using lemma 5.2 to show $f(j) < n^{\log n}$ for all j such that $1 \leq j \leq i$. Thus for all expanded nodes (j, y), $|y| < n^{\log n}$. Since we create unexpanded nodes only from expanded nodes, we can actually show for all nodes (j, y) of G, $|y| < n^{\log n}$. □

Lemma 5.2 *Suppose we have a function $f(j)$ with the following conditions:*

1. *$f(i) = n$ for some i, $1 \leq i \leq \log\log\log n$*
2. *$f(j-1) \leq f(j)^{\log\log\log f(j)}$ for all j, $1 \leq j \leq i$*

Then $f(j) < n^{\log n}$ for all j, $1 \leq j \leq i$.

Proof By induction on j. True for $j = i$ by assumption. Assume $f(k) < n^{\log n}$ for all k, $j < k \leq i$. We will show the lemma true for j.

For all k with $j < k \leq i$,

$$f(k-1) \leq f(k)^{\log\log\log f(k)} < f(k)^{\log\log\log(n^{\log n})} = f(k)^{g(n)}$$

for $g(n) = \log\log\log(n^{\log n}) = 1 + \log\log\log n$. Then we have

$$f(j) \leq f(i)^{g(n)^{i-j}} \leq n^{g(n)^{\log\log\log n}}$$

by repeatedly exponentiating $f(i) = n$ by $g(n)$ for $i - j$ times. To bound the exponent, we note

$$\log g(n)^{\log\log\log n} = \log\log\log n \log g(n) = \log\log\log n \log(1 + \log\log\log n) < \log\log n.$$

Thus $g(n)^{\log\log\log n} < \log n$ and thus $f(j) < n^{\log n}$ □

Since we restrict the length of y between n and $n^{\log n}$ and j between 1 and $\log\log\log n$, there can be only a finite number of nodes (j, y) of G.

5.4.6 Processing the Dependency Graph

After we create the dependency graph G for $M_i^A(x)$ as described above, we will process each expanded node (j, y) of G from smallest node to (i, x) in the order also described above, possibly changing G at each step.

As we process each node (j, y) of G we will either

1. Set enough of the oracle A to determine $M_j(y)$. We then encode $M_j(y)$ in A appropriately with its unset encoding strings. We remove node (j, y) from G, along with all its associated edges.
2. Make M_j improper. At this point we stop trying to encode $M_i(x)$, invoking the finite injury argument since $j \leq i$.

3. Put node (j, y) on *hold*. We will restructure the dependency graph so (j, y) only has edges to nodes (k, z) with $k > j$ and $|z| > |y|$.

As we process each node (j, y) of G all previously processed nodes not removed will be on hold. These held nodes depend solely on a larger index or input than (j, y). We will combine node (j, y) with all the held nodes it depends on into one single node. We will show the probabilistic machine corresponding to this node can not have too much power. This machine also depends solely on encodings of a larger index or input. We show we can apply one of the three actions above and then we process the next node. When we process node (i, x) it can not be put on hold because $|x| = n$ and no nodes of G have input of length less than n. We have then succeeded in encoding $M_i^A(x)$ or making it improper or making some machine with a smaller index improper.

When does a node (k, z) become unheld? The node (k, z) becomes unheld when we encode all of the nodes (k, z) depends on. However some of these nodes might themselves be put on hold. Suppose (k, z) depends on (j, y) and we decide to put (j, y) on hold, *i.e.* $M_k(z)$ depends on $M_j(y)$, which in turn depends on larger indices. Clearly then $M_k(z)$ depends only on what $M_j(y)$ depends on. Possibly (k, z) never becomes unheld to achieve the goal of encoding (i, x). We will keep the following invariant: When we process node (j, y), all held nodes only have edges to unprocessed nodes.

We now give a more precise description of how we process node (j, y):

1. Convert $M_j(y)$ to $M_j^*(y)$, a new machine that combines $M_j(y)$ with all the machines related to the held nodes (j, y) depends on. We describe the combining process below. We remove all edges from (j, y) to the held nodes. The machine $M_j^*(y)$ depends only on encodings relating to larger nodes.

2. We try to set the oracle to make $M_j^{*A}(y)$ accept with an improper probability.

3. If unsuccessful, we examine the influencing and simulating strings of $M_j^*(y)$. We argue that the simulating strings can not affect the output of $M_j^{*A}(y)$, and we remove the related edges from G. We set all the influencing strings of encodings of inputs at least as long as y to zero, also removing those edges from G.

4. If there are no more influencing strings then we have determined $M_j^{*A}(y)$. Properly encode the oracle. Recompute every machine related to a held node with an edge to (j, y). If we have now determined those machines, properly encode the oracle and remove those nodes from G. Finally, remove node (j, y) from G and all its remaining associated edges.

5. If influencing edges remain, we then place node (j, y) on hold. We combine every machine with a hold on (j, y) with $M_j^*(y)$ replacing edges to (j, y) with edges to the influencing nodes of (j, y).

We combine a machine M with a set of machines $\mathcal{M}$ as follows: We simulate M choosing the random coin tosses at random. When M asks an oracle query about the encoding of a machine $M' \in \mathcal{M}$, we would like to respond with whether M' accepts. We could simulate M' and respond to the oracle query with the output of M'. However with any given set of coin tosses, M may ask several oracle queries and each simulation could fail with probability

up to one third. Thus we could have a large probability of getting wrong answers on some of the oracle queries. Instead we simulate M' n times independently and take the majority answer, which gives us an exponentially small error. If we can show the number of oracle queries M can ask on any given set of coin tosses is much less than exponential then M will have a negligible probability of making a mistake on any computation path.

We call this combined machine M^*. If M runs in time $f(n)$ and the maximal running time of a machine in $\mathcal{M}$ is $g(n)$ then an upper bound on the time of M^* is $nf(n)g(f(n))$: A maximum of $f(n)$ oracle queries of length at most $f(n)$ each simulated n times.

Since we only do $O(\log\log\log n)$ steps of combining on any given machine we can show using similar calculations as lemma 5.2 that every M^* machine takes no more than $O(n^{\log n})$ steps for every node in the graph generated by (i,x) with $|x| = n$.

Define an *influencing string* of an oracle as an oracle query that appears on at least $2^{-\frac{n}{2}}$ of the paths.

Lemma 5.3 *If M runs in time $t(n)$ then for any given x of length n, $M(x)$ has at most $t(n)2^{\frac{n}{2}}$ influencing strings.*

Proof Suppose $M(x)$ has c computation paths. If $M(x)$ has more than $t(n)2^{\frac{n}{2}}$ influencing strings then these influencing strings account for more than $ct(n)$ oracle queries. However we can have at most $ct(n)$ oracle queries because at most $t(n)$ queries can be asked on each of the c computation paths. □

No $M_j^*(y)$ for (j,y) generated by (i,x) will have more than 2^n influencing strings for $n = |x|$.

Note no simulating string can affect the probability of a machines acceptance by more than $2^{-\frac{n}{2}}$. Thus if the output of a machine depends on settings of the simulating strings, then we can make the machine improper, because changing a single simulating string can not change a machine from accepting to rejecting or vice versa.

We need to show the oracle has room to encode $M_j^A(y)$. Let $m = |y|$. Recall we encode $M_j^A(y)$ in oracle strings of the form (j,y,r) where r can be any of the 2^{5m} strings of length $5m$. If we used any of these strings to make a machine improper then we set the remaining encoding strings to zero, invoking the finite injury argument. Notice the only other encoding strings of $M_j(y)$ that have been previously set are influencing strings from machines on inputs of length at most m. These are $\log\log\log m$ machines each asking at most 2^m influencing strings on 2^{m+1} inputs of size at most m. Influencing strings take up a grand total of at most $2^{2m+1}\log\log\log m$ of the encoding strings of (j,y,r) less than 2^{-2m} of the 2^{5m} encoding strings available. The noninfluencing strings consist of more than $1-2^{-2m} > \frac{2}{3}$ of the encoding strings available and thus we can properly encode whether $M_i^A(x)$ accepts.

5.4.7 Generalizing the Proof for All BPP Machines

We will give a sketch of the modifications of this proof necessary if we allow the probabilistic machines to base their oracle queries on answers to previous oracle queries. The computation paths on a machine M will now contain branches both for coin tosses and oracle queries. We create G using all possible branches of both types. When we work our way processing each node (j,y) of G we do the following:

1. We create $M^*(j,y)$ as before.

2. We will try all possible oracle settings of A to try to make $M^{*A}(j,y)$ improper. If we succeed then we longer need to continue processing G.

3. Look at the machine where it takes oracle query branches as though they were zero. We argue the simulating strings of this model can not affect the output of the original machine. As before, we set to zero influencing strings of an encoding of an input of length at least $|y|$. If there are other influencing strings, we will put the machine on hold and combine the appropriate machines.

4. We encode (j,y). In this case we may have introduced ones into A. We then recompute as well as combine all machines that had a hold on (j,y).

The remainder of the proof follows as before.

5.5 Other Results

Corollary 5.4 BPP = ZPTIME[n] = RTIME[n] $\subseteq$ NTIME[n] *for some oracle A.*

Proof Note in the proof of the main theorem we only introduce ones into the oracle when we make a machine improper or when we encode a machine. Except for a finite number of injuries, if $M_i^A(x)$ rejects then we will encode $M_i^A(x)$ entirely with zeros. A linear-time machine that picks an encoding string at random will never accept in this case so the oracle we created actually collapses BPP to RTIME[n]. If BPP = RTIME[n] then BPP = co-RTIME[n] and thus BPP = ZPTIME[n]. □

Corollary 5.5 *For some oracle A,* BPP *has linear size circuits.*

Proof Fix a probabilistic machine M_i. Look at all 2^n inputs of length n. For any (i,x,r) all but 2^{-2n} of the r's correctly encode $M_i^A(x)$. Thus there exists some r' such that (i,x,r') correctly encodes $M_i^A(x)$ for all x of length n. We can easily build a linear size circuit that determines if $(i,x,r') \in A$. □

Theorem 5.6 *For some oracle A,* $\Delta_2 \subseteq$ BPTIME[n].

Proof Let $M_1, M_2, \ldots$ be a list of Δ_2 machines, *i.e.* polynomial-time nondeterministic machines with access to a NP oracle. We set up the oracle A as in the proof of the main theorem but we encode our machines in a different way.

Look at the computation of $M_i(x)$ on undetermined oracle queries. Using techniques of Wilson [W] we note there exists a setting of polynomial many oracle questions that determines $M_i^A(x)$. We set the oracle in this way to determine $M_i^A(x)$. We then encode (i,x,r) properly. There are $\log\log\log n$ machines on 2^n inputs each requiring at most a polynomial number of oracle queries to be set. We have hardly used any of the 2^{5n} oracle questions available. Thus we will have no problem encoding $M_i^A(x)$. □

Note $\Delta_2 \not\subseteq$ RTIME[n] under any oracle, because $\Delta_2 \subseteq$ RTIME[n] would imply NP = NTIME[n], which contradicts Cook's result [C2].

Theorem 5.7 *For some oracle B, the class* BPTIME$[n]$ *does not contain* DTIME$[n^2]$.

Proof Let $M_1, M_2, \ldots$ be an enumeration of linear time probabilistic Turing machines. We can assume M_i runs in time at most in for inputs of length n. In step i we do the following:

Let $n = i + 1$. Set all strings of the oracle B of length less than n^2 to zero. Simulate M_i^B on input 1^n. The machine M_i^B can ask questions of length at most $in < n^2$ and thus is completely determined. If $M_i^B(1^n)$ accepts with probability less than 1/2 than we put 1^{n^2} in B otherwise we leave 1^{n^2} out of B.

Let the language $L = \{1^n | 1^{n^2} \in B\}$. A deterministic quadratic time machine with access to B can clearly accept L but the standard diagonalization argument shows no linear probabilistic time machine can accept L. □

Under this oracle B, BPTIME$[n]$ does not contain either BPP or Δ_2. We can use a similar argument to create oracles where NTIME$[n]$ does not contain BPP and BPP does not have linear size circuits.

Finally we will show the impossibility of simultaneously collapsing both BPP and NP to probabilistic linear time even though we can do either individually.

Theorem 5.8 *The following two statements can not both be true:*

$$\text{BPP} = \text{BPTIME}[n]$$
$$\text{NP} \subseteq \text{BPTIME}[n]$$

Proof Assume both statements are true. Then NP would be contained in BPP. By a result of Zachos [Z], NP $\subseteq$ BPP implies the entire polynomial-time hierarchy collapses to BPP and thus to BPTIME$[n]$. Then all languages in the polynomial-time hierarchy have n^4–size circuits which contradicts Kannan's result [K] that $\Sigma_2 \cap \Pi_2$ does not have n^k–size circuits for any fixed k. □

Note this proof relativizes; thus no oracle A exists that collapses both BPP and NP to probabilistic linear time even though we can collapse each individually.

Any complexity class that contains both NP and BPP can not be collapsed to BPTIME$[n]$. In particular, the set of all languages accepted by interactive proof systems must contain languages not recognizable in probabilistic linear time since IP contains both NP and BPP.

5.6 Conclusions and Further Research

This chapter shows the collapse of both BPP and Δ_2 to both BPTIME$[n]$ and linear size circuits with appropriate oracles. Also, $\Sigma_2 \cap \Pi_2$ can not be collapsed to either BPTIME$[n]$ or linear size circuits. Probabilistic linear time does not contain all languages accepted by interactive proof systems. However, it is unknown whether interactive proof systems can have linear size circuits. We believe an oracle exists under which all languages accepted by interactive proof systems have linear size circuits.

Ideally the questions in this paper should be resolved in the unrelativized world. This chapter shows solving such problems will be hard but not necessarily impossible. However it would likely require new techniques to solve them. We conjecture none of the collapses occur in the unrelativized world.

This thesis does not address the original question stated in section 5.1: Do interactive proof systems with a linear-time verifier accept a strictly smaller set of languages than interactive proof systems with a polynomial-time verifier? This question was the original motivation of the research of this chapter but has remained unsolved.

This chapter introduces several techniques for oracle construction. These methods may be useful in the construction of oracles for other problems.

This chapter gives an example of how probabilistic computation appears different than deterministic and nondeterministic computation. Perhaps there exist other results that may help to understand the differences and similarities in the nature of probabilistic computation and other types of computation such as interactive proof systems.

Bibliography

[AGH] Aiello, W., S. Goldwasser and J. Hastad, "On the power of Interaction", *Combinatorica*, to appear. Extended abstract available in *Proc. 27th FOCS*, 1986, pp. 368-379.

[AH] Aiello, W. and J. Hastad, "Statistical Zero-Knowledge Languages can be Recognized in Two Rounds", *JCSS*, to appear. Extended abstract available in *Proc. 28th FOCS*, 1987, pp. 439-448.

[AKLLR] Aleliunas, R., R. Karp, R. Lipton, L. Lovász, and C. Rackoff, "Random Walks, Universal Transversal Sequences, and the Complexity of Maze Problems", *Proc. 20th FOCS*, 1979, pp. 218-223.

[B] Babai, L., "Trading Group Theory for Randomness", *Proc. 17th STOC*, 1985, pp. 421-429.

[BM] Babai, L. and S. Moran, "Arthur-Merlin games: A Randomized Proof System, and a Hierarchy of Complexity Classes", *JCSS* **36** 2, 1988, pp. 254-276.

[BGS] Baker, T., J. Gill and R. Solovay, "Relativizations of the P = NP question", SIAM J. Comput., **4** 4, 1975, 431-442.

[BGKW] Ben-Or, M., S. Goldwasser, J. Kilian and A. Wigderson, "Multi-Prover Interactive Proofs: How to Remove Intractability Assumptions", *Proc. 20th STOC*, 1988, pp. 113-131.

[BG] Bennet C. and J. Gill, "Relative to a Random Oracle, $P^A \neq NP^A \neq$ co-NP^A with Probability One", SIAM J. Comput., **10**, 1981, pp. 96-113.

[BHZ] Boppana, R., J. Hastad and S. Zachos, "Does co-NP Have Short Interactive Proofs?", *Information Processing Letters*, **25** 2, 1987, pp. 127-132.

[BC] Brassard, G. and C. Crépeau, "Non-Transitive Transfer of Confidence: A Perfect Zero-Knowledge Interactive Protocol for SAT and Beyond", *Proc. 27th FOCS*, 1986, pp. 188-195.

[CW] Carter, J.L. and M.N. Wegman, "Universal Classes of Hash Functions", *JCSS* **18** 2, 1979, pp.143-154.

[Co1] Condon, A., *Computational Models of Games*, PhD Dissertation, Department of Computer Science, University of Washington at Seattle, 1987.

[Co2] Condon, A., "Space Bounded Probabilistic Game Automata", *proc. 3rd Structure in Complexity Theory Conf.*, 1988, pp. 162-174.

[CL] Condon, A., and R. Ladner, "Probabilistic Game Automata", *Proc. 1st Structure in Complexity Theory Conf.*, 1986, pp. 144-162.

[C1] Cook, S., "The Complexity of Theorem Proving Procedures", *Proc. 3rd STOC*, 1971, pp. 151-158.

[C2] Cook, S., "A hierarchy for Nondeterministic Time Complexity", *JCSS* **7** 4, 1973, pp.343-353.

[DH] Diffie, W. and M. Hellman, "New Directions in Cryptography", *IEEE Trans. on Inform. Theory* **IT-22** 6, 1976, pp. 644-654.

[E] Edmonds, J., "Paths, Trees and Flowers", *Canad. Jour. Math.* **17**, 1965, pp. 449-467.

[Fe] Feldman, P., "The Optimum Prover Lives in PSPACE", manuscript.

[FM] Feldman, P. and S. Micali, "Optimal Algorithms for Byzantine Agreement", *Proc. 20th STOC*, 1988, pp. 148-161.

[Fo] Fortnow, L., "The Complexity of Perfect Zero-Knowledge". In S. Micali, editor, *Randomness and Computation*, Volume 5 of *Advances in Computing Research*, JAI Press, 1988. Extended Abstract available in *Proc.* 19^{th} *Symposium on Theory of Computing*, 1987, pp. 204-209.

[FRS] Fortnow, L., J. Rompel and M. Sipser, "On the Power of Multi-Prover Interactive Protocols", *Proc.* 3^{rd} *Structure in Complexity Theory Conference*, 1988, pp. 156-161.

[FS1] Fortnow, L. and M. Sipser, "Are There Interactive Protocols for co-NP Languages?", *Information Processing Letters* **28**, North-Holland, 1988, pp. 249-251.

[FS2] Fortnow, L. and M. Sipser, "Probabilistic Computation and Linear Time", *Proc. 21st Symposium on Theory of Computing*, 1989, to appear.

[GJ] Garey, M. and D. Johnson, "Computers and Intractability: A Guide to the Theory of NP-Completeness", W.H. Freeman and Co., 1979.

[G] Gill, J., "Computation Complexity of Probabilistic Turing Machines", *SIAM J. Comp.* **6** 4, 1977, pp. 675-695.

[GMS] Goldreich, O., Y. Mansour and M. Sipser, "Interactive Proof Systems: Provers that never Fail and Random Selection", *Proc. 28th FOCS*, 1987, pp. 449-461.

[GMW1] Goldreich, O., S. Micali and A. Wigderson, "Proofs that Yield Nothing But their Validity and a Methodology of Cryptographic Protocol Design", *Proc. 27th FOCS*, 1986, pp. 174-187.

[GMW2] Godreich, O., S. Micali and A. Wigderson, "How to Play any Mental Game", *Proc. 18th STOC*, 1987, pp. 218-229.

[GMR] Goldwasser, S., S. Micali and C. Rackoff, "The Knowledge Complexity of Interactive Proof-Systems", *SIAM Journal on Computing* **18** 1, 1989, pp. 186-208. Extended abstract available in *Proc. 17th STOC*, 1985, pp. 291-304.

[GS] Goldwasser, S. and M. Sipser, "Private Coins versus Public Coins in Interactive Proof Systems". In S. Micali, editor, *Randomness and Computation*, Volume 5 of *Advances in Computing Research*, JAI Press, to appear. Extended Abstract available in *Proc. 18th STOC*, 1986, pp. 59-68.

[HaS] Hartmanis, J. and R. Stearns, "On the Computational Complexity of Algorithms", *Trans. AMS* **117**, 1965, pp. 285-306.

[HeS] Hennie, F. and R. Stearns, "Two-tape Simulation of Multitape Turing Machines", *JACM* **13** 4, 1966, pp. 533-546.

[HU] Hopcroft, J. and J. Ullman, "Introdcution to Automata Theory, Languages and Computation", Addison-Wesley, 1979.

[I] Immerman, I., "Nondeterministic Space is Closed Under Complement", *Proc. 3rd Structure in Complexity Theory Conf.*, 1988, pp. 112-115.

[K] Kannan, R., "Circuit-Size Lower Bounds and Nonreducibility to Sparse Sets", *Information and Control* **55** 1-3, 1982, pp. 40-56.

[Ka] Karp, R., "Reducibility among Combinatorial Problems", in R. Miller and J. Thatcher, eds. *Complexity of Computer Computations*, Plenum Press, 1972, pp. 85-103.

[KV] Karpinski, M. and R. Verbeek, "Randomness, Provability, and the Separation of Monte Carlo Time and Space", *LNCS* **270**, 1988, pp. 189-207.

[Ki] Kilian, J., *Uses of Randomness in Algorithms and Protocols*, PhD Dissertation, Massachusetts Institute of Technology, 1989.

[Ko] Ko, K., "Relativized Polynomial Time Hierarchies Having Exactly K Levels", *Proc. 20th STOC*, 1988, pp. 245-253.

[L] Ladner, R., "The Circuit Value Problem is Log Space Complete for P", *SIGACT News*, **7** 1, Jan. 1975, pp. 18-20.

[La] Lautemann, C., "BPP and the polynomial hierarchy", *Information Processing Letters* **17** 4, 1983, 215-217.

[NW] Nisan, N. and A. Wigderson, "Hardness vs. Randomness", *Proc. 29th FOCS*, 1988, pp. 2-11.

[PR] Peterson, G. and J. Reif, "Multiple-Person Alternation", *Proc. 20th FOCS*, 1979, pp. 348-363.

[R] Rackoff, C., "Relativized Questions Involving Probabilistic Algorithms", *Proc. 10th STOC*, 1978, pp. 338-342.

[RSA] Rivest, R., A. Shamir and L. Adleman, "A Method for Obtaining Digital Signatures and Public Key Cryptosystems", *CACM*, **21**, Feb. 1978, pp. 120-126.

[Sa] Santha, M., "Relativized Arthur-Merlin versus Merlin-Arthur Games", *Information and Computation* **80** 1, 1989, pp. 44-49.

[SFM] Seiferas, J., M. Fischer and A. Meyer, "Separating Nondeterministic Time Complexity Classes", *JACM* **25** 1, 1978, pp. 146-167.

[Si] Sipser, M., "A Complexity Theoretic Approach to Randomness", *Proc. 15th STOC*, 1983, pp. 330-335.

[SS] Solovay, S. and V. Strassen, "A Fast Monte-Carlo Test for Primality", *SIAM J. Comp.* **6**, 1977, pp.84-85.

[St] Stockmeyer, L., "The Polynomial-time Hierarchy", *Theoretical Computer Science* **3**, 1977, pp. 1-22.

[Su1] Sudborough, I., "Time and Tape Bounded Auxiliary Pushdown Automata", *Mathematical Foundations of Computer Science*, 1977, pp. 493-503.

[Su2] Sudborough, I., "On the Tape Complexity of Deterministic Context Free Languages", *JACM*, **25** 3, 1978, pp. 405-414.

[V] Venkateswaran, H., "Properties that Characterize LOGCFL", *Proc. 19th STOC*, 1987, pp. 141-150.

[W] Wilson, C., "Relativized Circuit Complexity", 24^{th} *FOCS*, 1983, pp. 329-334.

[Y] Yao, A., "Separating the Polynomial-Time Hierarchy by Oracles", *Proc. 26th FOCS*, 1985, pp. 1-10.

[Z] Zachos, S., "Probabilistic Quantifiers, Adversaries, and Complexity Classes: An Overview". In A. Selman, editor, *Proc. 1st Structure in Complexity Theory*, Volume 223 of *Lecture Notes in Computer Science*, Springer-Verlag, 1986, pp. 383-400.