Lecture 1

Review of Basic Computational Complexity

March 30, 2004
Lecturer: Paul Beame
Notes: Daniel Lowd

1.1 Preliminaries

1.1.1 Texts

There is no one textbook that covers everything in this course. Some of the discoveries are simply too recent. For the first portion of the course, however, two supplementary texts may be useful.

- Michael Sipser, *Introduction to the Theory of Computation*.
- Christos Papadimitrion, *Computational Complexity*

1.1.2 Coursework

Required work for this course consists of lecture notes and group assignments. Each student is assigned to take thorough lecture notes no more than twice during the quarter. The student must then type up the notes in expanded form in LaTeX. Lecture notes may also contain additional references or details omitted in class. A LaTeXstylesheet for this will soon be made available on the course webpage. Notes should be typed, submitted, revised, and posted to the course webpage within one week.

Group assignments will be homework sets that may be done cooperatively. In fact, cooperation is encouraged. Credit to collaborators should be given. There will be no tests.

1.1.3 Overview

The first part of the course will predominantly discuss complexity classes above NPrelate randomness, circuits, and counting, to P, NP, and the polynomial time hierarchy. One of the motivating goals is to consider how one might separate P from classes above P. Results covered will range from the 1970's to recent work, including tradeoffs between time and space complexity. In the middle section of the course we will cover the powerful role that interaction has played in our understanding of computational complexity and, in particular, the powerful results on probabilistically checkable proofs (PCP). These ideas were developed in the late 1980's continuing through the 1990's. Finally, we will look at techniques for separating non-uniform complexity classes from P. These have been applied particularly to separate classes inside P from P. The results here are from the 1980's and early 1990's.

1.2 Basic Complexity Classes

For this course we will use the following standard basis for defining complexity classes using multitape (offline) Turing machines.

Definition 1.1. An *(offline) Multitape Turing Machine* is a Turing machine that consists of

- Finite state control,
- A read-only input tape, and
- *Multiple* read-write storage tapes

(The read-only input tape will only be required for the definitions of space complexity.) Whenever we use the terminology Turing machine, or TM, we assume a multitape offline Turing machine unless otherwise specified; similarly for NTM in the nondeterministic case.

The two main parameters for measuring resources used by a computation are
Time = # of steps of the Turing machine, and
Space = largest numbered cell accessed on any storage tape.

Note that this is a small change from the definitions in the Sipser text which defines complexity classes in (a nonstandard way) using single tape Turing machines. Use of multiple tapes is important to distinguish between different subclasses in P. A single tape requires a Turing machine to move back and forth excessively, where a multitape Turing machine might be able to solve the problem much more efficiently.

Lemma 1.1. *For $T(n) \geq n$, if language A can be decided in time $T(n)$ by a multitape TM then a 1-tape TM can decide A in time $O(T^2(n))$ time.*

The following example shows that this is tight. Consider the language PALINDROME = $\{x \in \{0,1\}^* \mid x = x^R\}$. It is fairly straightforward to see how a multitape Turing machine could decide the language in linear time $O(|x|)$ by copying its input string in reverse onto a second tape and then comparing the two tapes character by character. It can also be shown that a single tape Turing machine must take longer:

Theorem 1.2 (Cobham). PALINDROME *requires time $\Omega(n^2)$ on 1-tape Turing machines.*

Proof. Exercise. □

One can simulate any multitape TM with only 2 tapes with a much smaller slowdown.

Lemma 1.3 (Hennie-Stearns). *For $T(n) \geq n$, if language A can be decided in time $T(n)$ by a multitape TM then a 1-tape TM can decide A in time $O(T(n) \log T(n))$ time.*

The proof of this fact is much more involved and we will look at this proof in a more general context.

1.2.1 Time Complexity

Definition 1.2. For $T : \mathbb{N} \to \mathbb{R}^+$,
$\mathsf{TIME}(T(n)) = \{A \subseteq \{0,1\}^* \mid A$ can be decided by a multitape TM M in $O(T(n))$ time$\}$, and
$\mathsf{NTIME}(T(n)) = \{A \subseteq \{0,1\}^* \mid A$ can be decided by a multitape NTM M in $O(T(n))$ time$\}$.

Definition 1.3. Recall the following derived time complexity classes,

$$\begin{aligned}
\mathsf{P} &= \bigcup_k \mathsf{TIME}(n^k) \\
\mathsf{NP} &= \bigcup_k \mathsf{NTIME}(n^k) \\
\mathsf{EXP} &= \bigcup_k \mathsf{TIME}(2^{n^k}) \\
\mathsf{NEXP} &= \bigcup_k \mathsf{NTIME}(2^{n^k}) \\
\mathsf{E} &= \bigcup_k \mathsf{TIME}(k^n) = \bigcup_k \mathsf{TIME}(2^{kn}) \\
\mathsf{NE} &= \bigcup_k \mathsf{NTIME}(k^n) = \bigcup_k \mathsf{NTIME}(2^{kn}).
\end{aligned}$$

Note that E and NE which we may encounter will be seen to be somewhat different from the others because they are not closed under polynomial-time reductions.

1.2.2 Polynomial-time Reductions

Many-one/Karp/Mapping reductions

Definition 1.4. $A \leq_m^p B$ iff there is a polynomial-time computable f such that $x \in A \Leftrightarrow f(x) \in B$

Turing/Cook/Oracle reductions

Definition 1.5. An *oracle TM* $M^?$ is an ordinary Turing machine augmented with a separate read/write *oracle query tape*, *oracle query state*, and 2 oracle answer states, Y and N. When furnished with an oracle B, whenever $M^?$ enters the oracle query state, and enters state Y or N depending on whether or not the contents of the oracle query tape is in language B. Cost for an oracle query is a single time step. If answers to oracle queries are given by membership in B, then we refer to the oracle TM as M^B.

Definition 1.6. $A \leq_T^p B$ iff there is a polynomial-time oracle TM $M^?$ such that $A = L(M^B)$

In other words, M can decide A in polynomial time given a subroutine for B that costs one time step per call.

1.2.3 NP-completeness

Definition 1.7. L is NP-*complete* iff:

1. $L \in \mathsf{NP}$

2. $\forall A \in \mathsf{NP}, A \leq_m^p L$

Theorem 1.4 (Cook). SAT = $\{\langle\varphi\rangle \mid \varphi$ *is a satisfiable propositional logic formula*$\}$ *is* NP*-complete.*

Definition 1.8. For any complexity class C, define co$\mathsf{C} = \{L \mid \overline{L} \in \mathsf{C}\}$.

For example, the class coNP is the set of all languages whose complements are in NP. The following languages are both coNP-complete:

- UNSAT = $\{\langle\varphi\rangle \mid \phi$ is an unsatisfiable propositional logic formula $\}$
- TAUT = $\{\langle\varphi\rangle \mid \varphi$ is a propositional logic tautology $\}$

Note: UNSAT $\leq_T^P$ SAT since Turing reductions don't distinguish between languages and their complements.

Definition 1.9. Define $\forall^k$ and $\exists^k$ as quantifiers over $\{0,1\}^{\leq k}$ the set of binary strings of length at most k.

Using this notation we have an alternative characterization of NP in terms of polynomial-time verifiers.

$A \in \mathsf{NP} \Leftrightarrow$ there is some $R \in \mathsf{P}$ and polynomial $p : \mathbb{N} \to \mathbb{N}$ such that $A = \{x \mid \exists^{p(|x|)} y \cdot (x,y) \in R\}$; or in functional rather than set form there is some polynomial-time computable R such that $A = \{x \mid \exists^{p(|x|)} y \cdot R(x,y)\}$.

$A \in$ coNP $\Leftrightarrow$ there is some $R \in \mathsf{P}$ and polynomial $p : \mathbb{N} \to \mathbb{N}$ such that $A = \{x \mid \forall^{p(|x|)} y \cdot (x,y) \in R\}$.

1.2.4 Space Complexity

Definition 1.10. For $S : \mathbb{N} \to \mathbb{R}^+$, define
SPACE$(S(n)) = \{A \subseteq \{0,1\}^* \mid A$ is decided by a multitape (offline) TM using storage $O(S(n))\}$
NSPACE$(S(n)) = \{A \subseteq \{0,1\}^* \mid A$ is decided by a multitape (offline) NTM using storage $O(S(n))\}$

Definition 1.11.

$$\begin{aligned} \mathsf{PSPACE} &= \bigcup_k \mathsf{SPACE}(n^k) \\ \mathsf{L} &= \mathsf{SPACE}(\log n) \\ \mathsf{NL} &= \mathsf{NSPACE}(\log n) \end{aligned}$$

Theorem 1.5. *For* $S(n) \geq \log n$*,*
(a) [Savitch] NSPACE$(S(n)) \subseteq$ SPACE$(S^2(n))$*,*
(b) [Immerman-Szelepcsenyi] NSPACE$(S(n)) =$ co $-$ NSPACE$(S(n))$.

Savitch's theorem implies that PSPACE could equally well have been defined using nondeterministic space complexity.

Theorem 1.6. *For* $S(n) \geq \log n$*,* NSPACE$(S(n)) \subseteq$ TIME$(2^{O(S(n))})$.

Proof idea. An NSPACE$(S(n))$ computation accepts an input x iff there is a computation path from the starting configuration s to a unique accepting configuration t in the graph of all possible configurations with input x, which there are $2^{O(S(n))}$ nodes. This can be solved in time linear in the size of the graph using BFS or DFS. □

Theorem 1.7. $\mathsf{NTIME}(T(n)) \subseteq \mathsf{SPACE}(T(n))$.

Proof idea. Each computation of the NTM of length $cT(n)$ visits at most $cT(n)+1$ cells on each storage tape. The algorithm successively generates all possible sequences of nondeterministic choices of length $cT(n)$ and exhaustively tries each sequence. □

Corollary 1.8. $\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \begin{matrix}\mathsf{NP}\\ \mathsf{coNP}\end{matrix} \subseteq \mathsf{PSPACE} \subseteq \mathsf{EXP} \subseteq \mathsf{NEXP}$.

Definition 1.12. A language L is PSPACE-*complete* iff

1. $L \in \mathsf{PSPACE}$, and
2. $\forall A \in \mathsf{PSPACE}$, $A \leq_m^p L$.

Definition 1.13. Define
TQBF = $\{\langle\Psi\rangle \mid \exists k, Q_1, Q_2, \ldots, Q_k \in \{\exists, \forall\}$ such that $\Psi = Q_1 x_1 \ldots Q_k x_k \varphi$, where φ is a propositional logic formula in $x_1, x_2, \ldots, x_n$, and Ψ evaluates to true $\}$.

Theorem 1.9. TQBF *is* PSPACE-*complete.*

1.2.5 Complexity Class Hierarchy Theorems

Definition 1.14. $T(n)$ is *time-constructable* iff there is a TM running in time $\leq T(n)$ that computes $1^n \to T(n)$, where $T(n)$ is expressed in binary.
$S(n)$ is *space-constructable* iff there is a TM running in space $\leq S(n)$ that computes the function $1^n \to S(n)$, where $S(n)$ is expressed in binary.

Theorem 1.10. *1. If $g(n) \geq \log(n)$ is space-constructable and $f(n)$ is $o(g(n))$ then* $\mathsf{SPACE}(f(n)) \subsetneq \mathsf{SPACE}(g(n))$, *and* $\mathsf{NSPACE}(f(n)) \subsetneq \mathsf{NSPACE}(g(n))$.

2. If $g(n) \geq n$ is time constructable and $f(n)\log f(n)$ is $o(g(n))$ then $\mathsf{TIME}(f(n)) \subsetneq \mathsf{TIME}(g(n))$.

Proof Sketch. The general idea of all of these theorems is to diagonalize over all TM's with time (space) bounded by $f(n)$. In order to do this one diagonalizes over all Turing machines but with an extra clock (in the case of time) or space limiter (in the case of space) to make sure that the machine involved won't take too many resources.

In the case of the space hierarchy, this diagonalization must be done by a single Turing machine so in order to simulate other Turing machines with larger alphabets there must be a constant-factor slowdown in the simulation. The Immerman-Szelepcsenyi theorem allows the complementation to take place in the case of nondeterministic space.

For the time hierarchy the proof in the multitape case is very different from the single tape case given in Sipser's text despite the fact that the claim is the same. In the single tape case, the $O(\log f(n))$ factor is due to the requirement to update the clock (and shift it along the tape) at each step. In the multitape case, the $O(\log f(n))$ factor slowdown is due the requirement of having a machine with a fixed number of tapes simulate machines with arbitrary numbers of tapes. □

Corollary 1.11. $\mathsf{NL} \subsetneq \mathsf{PSPACE}$; $\mathsf{P} \subsetneq \mathsf{EXP}$.

1.2.6 Relativized Complexity Classes

Definition 1.15. Given a language A, we can define $\mathsf{P}^A = \{B \mid B \leq_T^p A\}$.

That is, to obtain the languages in P^A we can take any polynomial-time oracle TM $M^?$, plug in A as the oracle and look at what language the machine accepts. This yields an alternate version of the definition that can also be extended to nondeterministic Turing machines.

Definition 1.16. For any language A define $\mathsf{P}^A = \{L(M^A) \mid M^?$ is a polynomial-time oracle TM $\}$ and $\mathsf{NP}^A = \{L(M^A) \mid M^?$ is a polynomial-time oracle NTM $\}$.
For any complexity class C define $\mathsf{P}^{\mathsf{C}} = \bigcup_{A \in \mathsf{C}} \mathsf{P}^A$, $\mathsf{NP}^{\mathsf{C}} = \bigcup_{A \in \mathsf{C}} \mathsf{NP}^A$,

Remark. Note that $\mathsf{P}^{\mathrm{SAT}} = \mathsf{P}^{\mathsf{NP}}$ by the fact that SAT is NP-complete under polynomial-time Turing reductions.

Lecture 2

Polynomial Time Hierarchy and Nonuniform Complexity

April 1, 2004
Lecturer: Paul Beame
Notes: Ioannis Giotis

2.1 Definition

Recall the following definitions of relativized complexity classes.

Definition 2.1. For any language A define $\mathsf{P}^A = \{L(M^A) \mid M^?$ is a polynomial-time oracle TM $\}$ and $\mathsf{NP}^A = \{L(M^A) \mid M^?$ is a polynomial-time oracle NTM $\}$.
For any complexity class C define $\mathsf{P}^{\mathsf{C}} = \bigcup_{A\in\mathsf{C}} \mathsf{P}^A$, $\mathsf{NP}^{\mathsf{C}} = \bigcup_{A\in\mathsf{C}} \mathsf{NP}^A$,

Definition 2.2. We define the classes in the polynomial-time hierarchy by

$$\begin{aligned}
\Sigma_0\mathsf{P} &= \Pi_0\mathsf{P} = \Delta_0\mathsf{P} = \mathsf{P} \\
\Delta_{i+1}\mathsf{P} &= \mathsf{P}^{\Sigma_i\mathsf{P}} \\
\Sigma_{i+1}\mathsf{P} &= \mathsf{NP}^{\Sigma_i\mathsf{P}} \\
\Pi_{i+1}\mathsf{P} &= \mathsf{coNP}^{\Sigma_i\mathsf{P}}
\end{aligned}$$

Unwinding the definition we obtain some basic complexity classes.

$$\begin{aligned}
\Delta_1\mathsf{P} &= \mathsf{P}^{\mathsf{P}} = \mathsf{P} \\
\Sigma_1\mathsf{P} &= \mathsf{NP}^{\mathsf{P}} = \mathsf{NP} \\
\Pi_1\mathsf{P} &= \mathsf{coNP}^{\mathsf{P}} = \mathsf{coNP} \\
\Delta_2\mathsf{P} &= \mathsf{P}^{\mathsf{NP}} = \mathsf{P}^{\mathrm{SAT}} \supseteq \mathsf{coNP} \\
\Sigma_2\mathsf{P} &= \mathsf{NP}^{\mathsf{NP}} \\
\Pi_2\mathsf{P} &= \mathsf{coNP}^{\mathsf{NP}}
\end{aligned}$$

An example language in P^{NP} is the following, which is the difference between $\{\langle G,k\rangle \mid G$ has a k-clique$\}$ and $\{\langle G,k\rangle \mid G$ has a $(k+1)$-clique$\}$.

$$\textsc{Exact-Clique} = \{\langle G,k\rangle \mid \text{the largest clique in } G \text{ has size } k\}$$

Observe that, since oracle TMs can easily flip the answers they receive from the oracle, $\mathsf{P}^{\Sigma_i\mathsf{P}} = \mathsf{P}^{\Pi_i\mathsf{P}}, \mathsf{NP}^{\Sigma_i\mathsf{P}} = \mathsf{NP}^{\Pi_i\mathsf{P}}, \mathsf{coNP}^{\Sigma_i\mathsf{P}} = \mathsf{coNP}^{\Pi_i\mathsf{P}}$

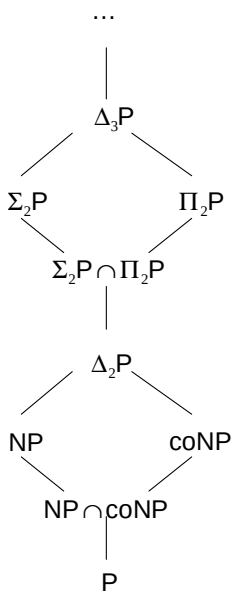

Figure 2.1: The polynomial time hierarchy

Definition 2.3. The *polynomial-time hierarchy* $\mathsf{PH} = \bigcup_k \Sigma_k\mathsf{P} = \bigcup_k \Pi_k\mathsf{P}$

Probably the most important question about the polynomial-time hierarchy is given by the following conjecture.

Conjecture 2.1. $\forall k\ \mathsf{PH} \neq \Sigma_k\mathsf{P}$.

Note that this very natural conjecture generalizes the conjectures that $\mathsf{P} \neq \mathsf{NP}$, since if $\mathsf{P} = \mathsf{NP}$ then $\mathsf{PH} = \mathsf{P} = \Sigma_0\mathsf{P}$, and that $\mathsf{NP} \neq \mathsf{coNP}$, since in that case $\mathsf{PH} = \mathsf{NP} = \Sigma_1\mathsf{P}$. Several results in this course will be conditional based on this conjecture, namely that some natural property will hold unless the polynomial-time hierarchy PH collapses to some level, say $\Sigma_k\mathsf{P}$.

2.2 Alternative Characterization of the Hierarchy Classes

The following characterization of languages in $\Sigma_k\mathsf{P}$ is useful for obtaining a simpler alternative characterization of PH.

Theorem 2.2. *$L \in \Sigma_{i+1}\mathsf{P}$ if and only if there exists a language $R \in \Pi_i\mathsf{P}$ and polynomial $p : \mathbb{N} \to \mathbb{N}$ such that $L = \{x|\ \exists^{p(|x|)}y.\ (x, y) \in R\}$.*

Proof. By induction. Base case $i = 0$ follows from the alternate characterization of NP: $L \in \mathsf{NP} \Leftrightarrow$ there is some $R \in \mathsf{P}$ and polynomial $p : \mathbb{N} \to \mathbb{N}$ such that $L = \{x \mid \exists^{p(|x|)}y.\ (x, y) \in R\}$

For the induction step, $i > 0$ let $L \in \Sigma_{i+1}\mathsf{P}$. We will show how to build a certificate y and a relation $R \in \Pi_i\mathsf{P}$ suitable for the characterization. By definition

$$L \in \mathsf{NP}^{\Sigma_i\mathsf{P}} = \mathsf{NP}^{\mathsf{NP}^{\Pi_{i-1}\mathsf{P}}}$$

which is well-defined since $i - 1 \geq 0$. This means that there is a polynomial-time oracle NTM $M^?$ such that if $M^?$ is given an oracle for a $\Sigma_i P$ set A, $L(M^A) = L$.

We consider the various parts of the computation on input x for L. $M^?$ has nondeterministic moves so part of the certificate y for x will consist of the nondeterministic moves of $M^?$; the certificate y will also contain the values of all the answers that $M^?$ receives from the oracle for A. Let these parts of y be $\tilde{y}$. Given $\tilde{y}$ in polynomial time we can check that the computation of $M^?$ given the oracles answers could follow the nondeterministic moves and accept.

The problem is that we don't know if the oracle answers purported to be according to A are correct. Given that the computation of $M^?$ is consistent with $\tilde{y}$, we have fixed the polynomially many oracle queries $z_1 \ldots, z_m$, say, that will be made to the oracle for A. The rest of the certificate y will be certificates that each of the answers given for A on these strings is actually correct.

We can verify each of the *yes* answers to A as follows: By applying the inductive hypothesis to A there exists a set $R' \in \Pi_{i-1}\mathsf{P}$ (and a polynomial q) such that

$$z_i \in A \Leftrightarrow \exists^{q(|x|)} y_i.(z_i, y_i) \in R'$$

If the answer to $z_i \in A$ is *yes* then we include this y_i in the certificate y. Since $R' \in \Pi_{i-1}\mathsf{P} \subseteq \Pi_i\mathsf{P}$ the algorithm for R can simply check that $(z_i, y_i) \in R'$ for each query z_i to A that is answered *yes*.

If the answer to whether or not $z_i \in A$ is *no* then $z_i \in \overline{A} \in \Pi_i\mathsf{P}$. Thus the new $\Pi_i\mathsf{P}$ machine R will check $\tilde{y}$, y_i if the answer was *yes*, and z_i directly if the answer for z_i was *no*. □

Corollary 2.3. *$L \in \Pi_{i+1}\mathsf{P}$ if and only if there is a relation $R \in \Sigma_{i+1}\mathsf{P}$ and polynomial $p : \mathbb{N} \to \mathbb{N}$ such that $L = \{x|\ \forall^{p(|x|)} y.\ (x, y) \in R\}$.*

Corollary 2.4. *$L \in \Sigma_i\mathsf{P}$ if and only if there is a polynomial-time computable set R and polynomial $p : \mathbb{N} \to \mathbb{N}$ such that*

$$L = \{x|\ \exists^{p(|x|)} y_1 \forall^{p(|x|)} y_2 \cdots Q y_i.\ (x, y_1, y_2, \ldots, y_i) \in R\}$$

where $Q = \begin{cases} \exists^{p(|x|)} & \text{if } i \text{ is odd,} \\ \forall^{p(|x|)} & \text{if } i \text{ is even.} \end{cases}$

2.2.1 Some $\Sigma_i\mathsf{P}$- and $\Pi_i\mathsf{P}$-complete Problems

Definition 2.4. Define
$\Sigma_i\mathrm{TQBF} = \{\langle\psi\rangle|\ \psi \in \mathrm{TQBF}$ and the quantifiers of ψ are of the form $\overrightarrow{\exists} y_1 \overrightarrow{\forall} y_2 \cdots Q y_k \psi Q\}$;
i.e. there are k groups of alternating quantifiers beginning with $\exists$. Similarly we can define $\Pi_i\mathrm{TQBF}$ with k groups of alternations beginning with $\forall$.

Theorem 2.5. *For $i \geq 1$, $\Sigma_i\mathrm{TQBF}$ is $\Sigma_i\mathsf{P}$-complete and $\Pi_i\mathrm{TQBF}$ is $\Pi_i\mathsf{P}$-complete.*

Proof Sketch. We apply the characterizations of $\Sigma_i\mathsf{P}$ and $\Pi_i\mathsf{P}$ from Corollary 2.4. The quantifiers match up perfectly except that the polynomial-time computable set R has been replaced by a simple Boolean formula. We cannot in general replace polynomial-time computation by Boolean formulas however, we can replace it in a way that is similar to the tableau from Cook's proof of the NP-completeness of SAT.

First suppose that the last quantifier for the complexity class is an $\exists$ quantifier. We can add additional Boolean variables z that represent the internal states of the computation of R and create a formula φ such that $(x, y_1, \ldots, y_i) \in R$ if and only if $\exists z \varphi(x, y_1, \ldots, y_i, z)$. Since the last quantifier was already an $\exists$ quantifier we can append z to it and simulate the formula.

If the last quantifier for the complexity class is a $\forall$ quantifier then we use the same internal variables z except that the formula for R is now $\forall z(\varphi'(x, y_1, \ldots, y_i, z) \rightarrow \varphi''(z))$ where φ' ensures that z is a correct computation on input $(x, y_1, \ldots, y_i)$ and φ'' ensures that z is accepting. In this case the $\forall z$ merges with the last $\forall$ in the alternation. □

2.2.2 Hierarchy Collapse

Lemma 2.6. *If* $\Sigma_k\mathsf{P} = \Pi_k\mathsf{P}$ *then* $\mathsf{PH} = \Sigma_k\mathsf{P} \cap \Pi_k\mathsf{P}$

Proof. We'll show that assumption implies that $\Sigma_{k+1}\mathsf{P} = \Sigma_k\mathsf{P}$ which will suffice. Let $A \in \Sigma_{k+1}\mathsf{P}$. Therefore, there is some polynomial-time R and polynomial p such that

$$A = \{x|\ \exists^{p(|x|)} y_1 \quad \underbrace{\forall^{p(|x|)} y_2 \cdots Q y_{k+1}.\ (x, y_1, y_2, \ldots, y_{k+1}) \in R}_{\in \Pi_k\mathsf{P} = \Sigma_k\mathsf{P}} \quad \}.$$

Therefore there is some polynomial-time relation R' and polynomial p' such that A can be expressed as

$$A = \{x| \quad \underbrace{\exists^{p(|x|)} y_1 \exists^{p'(|x|)} y_1'}_{\exists^{p''(|x|)}(y_1, y_1')} \quad \forall^{p'(|x|)} y_2' \cdots Q^{p'(|)} y_k'.\ (x, y_1, y_1', y_2', \ldots, y_k') \in R'\}$$

From which it follows that $A \in \Sigma_k\mathsf{P}$. □

If this happens we say that PH collapses to the k-th level.

2.2.3 A Natural Problem in $\Pi_2\mathsf{P}$

Definition 2.5. Let MINCIRCUIT $= \{\langle C\rangle|\ C$ is a circuit that is not equivalent to any smaller circuit$\}$.

Note that $\langle C\rangle \in$ MINCIRCUIT $\Leftrightarrow \forall\langle D\rangle, size(D) < size(C), \exists y$ s.t. $D(y) \neq C(y)$ Thus MINCIRCUIT is in $\Pi_2\mathsf{P}$. It is still open if it is in $\Sigma_2\mathsf{P}$ or if it is $\Pi_2\mathsf{P}$-complete.

2.3 Non-uniform Complexity

The definitions of Turing machines yield finite descriptions of infinite languages. These definitions are *uniform* in that they are fixed for all input sizes. We now consider some definitions of *non-uniform* complexity classes.

2.3.1 Circuit Complexity

Definition 2.6. Let $\mathbb{B}_n = \{f|\ f : \{0,1\}^n \rightarrow \{0,1\}\}$. A *basis* Ω is defined as a finite subset of $\bigcup_n \mathbb{B}_n$.

Definition 2.7. A *Boolean circuit over basis* Ω is a finite directed acyclic graph C each of whose nodes is

- a source node labelled by either an *input variable* in $\{x_1, x_2, \ldots\}$ or *constant* $\in \{0,1\}$, or

- a node of in-degree $d > 0$ labeled by a *gate* function $g \in \mathbb{B}_d \cap \Omega$,

and which has one designated *output node* (gate). A circuit C has two natural measures of complexity, its *size*, $size(C)$, which is the number of gates in C and its *depth*, $depth(C)$ which is the length of the longest path from an input (source) to the output node of C.

Typically the elements of Ω we use are symmetric and unless otherwise specified we will assume the so-called De Morgan basis $\Omega = \{\wedge, \vee, \neg\} \subseteq \mathbb{B}_1 \cup \mathbb{B}_2$.

Definition 2.8. C is *defined on* $\{x_1, x_2, \ldots, x_n\}$ if its input variables are contained in $\{x_1, x_2, \ldots, x_n\}$. C defined on $\{x_1, x_2, \ldots, x_n\}$ computes a function $f \in \mathbb{B}_n$ in the obvious way.

Definition 2.9. A *circuit family* C is an infinite sequence of circuits $\{C_n\}_{n=0}^{\infty}$ such that C_n is defined on $\{x_1, x_2, \ldots, x_n\}$.

Circuit family C has size $S(n)$, depth $d(n)$, iff for each n

$$\begin{aligned} size(C_n) &\leq S(n) \\ depth(C_n) &\leq d(n). \end{aligned}$$

Circuit family C *decides* or *computes* a language $A \subseteq \{0,1\}^*$ iff for every input $x \in \{0,1\}^*$, $C_{|}x|(x) = 1 \Leftrightarrow x \in A$.

Definition 2.10. We say $A \in \mathsf{SIZE}(S(n))$ if there exists a circuit family over the De Morgan basis of size $S(n)$ that computes A. Similarly we define $A \in \mathsf{DEPTH}(d(n))$.

$$\mathsf{POLYSIZE} = \bigcup_k \mathsf{SIZE}(n^k + k)$$

Our definitions of size and depth complexity classes somewhat arbitrarily care about constant factors. For Turing machines we are stuck with them because alphabet sizes are variables but for circuits the complexity does not involve such natural constant factors. (We may regret decision this later!)

Remark. $\exists A \in \mathsf{POLYSIZE}$ such that A is not decidable.

Lecture 3

Randomized Computation and the Polynomial-time Hierarchy

April 4, 2004
Lecturer: Paul Beame
Notes: Ethan Phelps-Goodman

3.1 Undecidable Languages in POLYSIZE

By the proof of the Cook-Levin theorem we know that all languages in P have polynomial size circuits. The converse is not true, however. In fact, there are languages with polynomial size circuits that are not decidable. For example,

$$A = \{1^{\langle M,x\rangle} \mid \text{Turing machine } M \text{ halts on input } x\} \in \mathsf{SIZE}(n)$$

where $1^{\langle M,x\rangle}$ denotes the unary encoding of the binary representation of machine M and input x. The construction of the circuit is as follows: Each input size corresponds to a particular machine and input. If the machine would halt on this input, then the circuit consists of the AND of all input variables. If the machine does not halt on the input then the circuit's output gate is always 0. This seemingly strange situation arises from the way we have defined circuit families. In particular, you are allowed to use an unbounded amount of computation to construct each particular circuit. This property is called non-uniformity.

(In a uniform circuit family each circuit can be built (or the structure of the gates in its underlying graph can be decided) in some small time-bounded or space-bounded class. There are different notions of uniform circuit classes depending precisely on what notion is used. Common examples used are log-space, or polynomial time.)

In fact, even constant-size circuits can decide undecidable languages: For example,

$$A = \{x \in \{0,1\}^* \mid \text{the } |x|^{th} \text{ TM } M_{|x|} \text{ halts on input } 1^{|x|}\} \in \mathsf{SIZE}(1).$$

3.2 Turing Machines with Advice

Last lecture introduced non-uniformity through circuits. An alternate view of non-uniformity uses Turing machines with an advice tape. The advice tape contains some extra information that depends only on the length of the input; i.e., on input x, the TM gets $(x, \alpha_{|x|})$.

Definition 3.1. $\mathsf{TIME}(T(n))/f(n) = \{A \mid A \text{ is decided in time } O(T(n)) \text{ by a TM with advice sequence } \{\alpha_n\}_{n=0}^{\infty} \text{ such that } |\alpha_n| \text{ is } O(f(n))\}$.

Now we can define the class of languages decidable in polynomial time with polynomial advice:

Definition 3.2. $\mathsf{P/poly} = \bigcup_{k,l} \mathsf{TIME}(n^k)/n^l$

Lemma 3.1. $\mathsf{P/poly} = \mathsf{POLYSIZE}$

Proof.
$\mathsf{POLYSIZE} \subseteq \mathsf{P/poly}$: Given a circuit family in $\mathsf{POLYSIZE}$, produce a TM with advice M that interprets its advice string as the description of a circuit and evaluates that circuit on the input. Use $\{\langle c_n \rangle\}_{n=0}^{\infty}$ as the advice strings. These are polynomial size and the evaluation is polynomial time.

$\mathsf{P/poly} \subseteq \mathsf{POLYSIZE}$: Given a TM M and a sequence of advice strings $\{\alpha_n\}_{n=0}^{\infty}$, use the tableau construction from the proof of the Cook-Levin Theorem to construct a polynomial-size circuit family with the advice strings hard-coded into the input. □

Typically, people tend to use the equivalent $\mathsf{P/poly}$ rather than $\mathsf{POLYSIZE}$ to describe the complexity class since it emphasizes the fact that it is the natural non-uniform generalization of P.

3.3 Probabilistic Complexity Classes

Definition 3.3. A *probabilistic Turing machine* is an NTM where each configuration has exactly one or two legal next moves. Steps with two moves are called *coin flip* steps. By viewing each such step as the flip of a fair coin we define the *probability* of a branch b being executed is $\Pr[b] = 2^{-k}$, where k is the number of coin flip steps along branch b. Then we define the probability of acceptance as:

$$\Pr[M \text{ accepts } w] = \sum_{\text{branches } b \text{ on which } M \text{ accepts } w} \Pr[b]$$

and $\Pr[M \text{ rejects } w] = 1 - \Pr[M \text{ accepts } w]$.

Probabilistic TMs can also be viewed as ordinary multi-tape TMs with an extra read-only coin flip tape. If r is the string of coin flips and machine M runs in time $T(n)$ then $|r| \leq T(n)$. Now we can write the answer of M on x as a function $M(x, r)$ which equals 1 if M accepts x given random string r.

We can now define probabilistic complexity classes.

Definition 3.4. Randomized Polynomial Time: A language $A \in \mathsf{RP}$ iff there exists a probabilistic polynomial time TM M such that for some $\epsilon < 1$,

1. $\forall w \in A$, $\Pr[M \text{ accepts } w] \geq 1 - \epsilon$.
2. $\forall w \notin A$, $\Pr[M \text{ accepts } w] = 0$.

The error, ϵ, is fixed for all input size. $\mathsf{RP}$is the class of problems with one-sided error (ie. an accept answer is always correct, whereas a reject may be incorrect.) We can also define coRP, which has one-sided error in the other direction. The following class encompasses machines with two-sided error:

Definition 3.5. Bounded-error Probabilistic Polynomial Time: A language $A \in \mathsf{BPP}$ iff there exists a probabilistic TM M running in polynomial time such that for some $\epsilon < 1/2$:

1. $\forall w \in A$, $\Pr[M \text{ accepts } w] \geq 1 - \epsilon$
2. $\forall w \notin A$, $\Pr[M \text{ rejects } w] \geq 1 - \epsilon$

We will slightly abuse notation and conflate languages and their characteristic functions; i.e., for a language A,

$$A(w) = \begin{cases} 1 & \text{if } w \in A \\ 0 & \text{if } w \notin A. \end{cases}$$

Using this we can say that $A \in \mathsf{BPP}$ iff there is some probabilistic polynomial-time TM M such that

$$\Pr[M(w, r) = A(w)] \geq 1 - \epsilon.$$

We will also define a zero-sided error complexity class:

Definition 3.6. Zero-error Probabilistic Polynomial Time: $\mathsf{ZPP} = \mathsf{RP} \cap \mathsf{coRP}$

The motivation for this terminology is the following lemma.

Lemma 3.2. *If $A \in \mathsf{ZPP}$ then there is a probabilistic TM M such that $L(M) = A$ and the expected running time of M is polynomial.*

Proof. Let M_1 be an RP machine for A, and M_2 be a coRP machine for A; i.e., an RP machine for $\overline{A}$. M repeatedly runs M_1 and M_2 alternately until one accepts. If M_1 accepts, then accept. If M_2 accepts then reject. Let $\epsilon = \max\{\epsilon_1, \epsilon_2\}$. We expect to have to run $\frac{1}{1-\epsilon}$ trials before one accepts. Thus M decides A in polynomial expected time. □

The last probabilistic complexity class is much more powerful:

Definition 3.7. Probabilistic Polytime: $A \in \mathsf{PP}$ iff there is a probabilistic polynomial time TM M such that

$$\Pr[M(w, r) = A(w)] > 1/2.$$

Here the error is allowed depend on the input size and be exponentially close to $1/2$.

Remark. Running a polynomial time experiment using a machine witnessing that $A \in \mathsf{PP}$ will in general not be enough to tell whether or not an input $x \in A$ or not because the difference between acceptance and rejection probabilities may be exponentially close to 1.

3.4 Amplification

The following lemma shows that in polynomial time we can reduce errors that are polynomially close to $1/2$ to exponentially small values.

Lemma 3.3. *Let M be a probabilistic TM with two-sided error $\epsilon = 1/2 - \delta$ running in time $T(n)$. Then for any $m > 0$ there is a probabilistic polytime TM M' with runtime at most $O(\frac{m}{\delta^2} T(n))$ and error at most 2^{-m}.*

Proof. M' simply runs M some number k times on independently chosen random strings and takes the majority vote of the answers. For convenience we assume that k is even. The error is:

$$\begin{aligned}
\Pr[M'(x) \neq A(x)] &= \Pr[\geq k/2 \text{ wrong answers on } x] \\
&= \sum_{i=0}^{k/2} \Pr[k/2+i \text{ wrong answers of } M \text{ on } x] \\
&\leq \sum_{i=0}^{k/2} \binom{k}{k/2+i} \epsilon^{k/2+i}(1-\epsilon)^{k/2-i} \\
&\leq \sum_{i=0}^{k/2} \binom{k}{k/2+i} \epsilon^{k/2}(1-\epsilon)^{k/2} \qquad \text{since } \epsilon \leq 1-\epsilon \text{ for } \epsilon \leq 1/2 \\
&\leq 2^k \epsilon^{k/2}(1-\epsilon)^{k/2} \\
&= [4(1/2-\delta)(1/2+\delta)]^{k/2} \\
&= (1-4\delta^2)^{k/2} \\
&\leq e^{-2\delta^2 k} \qquad \text{since } 1-x \leq e^{-x} \\
&\leq 2^{-m} \qquad \text{for } k = m/\delta^2.
\end{aligned}$$

□

A similar approach can be used with an RP language, this time accepting if any of the k trials accept. This gives an error of ϵ^k, where we can choose $k = \frac{m}{\log(\frac{1}{\epsilon})}$.

3.5 Randomness vs. Advice

The following theorem show that randomness is no more powerful than advice in general.

Theorem 3.4 (Gill, Adleman). $\mathsf{BPP} \subseteq \mathsf{P/poly}$

Proof. Let $A \in \mathsf{BPP}$. By the amplification lemma, there exists a BPP machine M for A and a polynomial bound p such that for all $x \in \{0,1\}^n$,

$$\Pr_{r \in \{0,1\}^{p(n)}}[M(x,r) \neq A(x)] \leq 2^{-2n}$$

For $r \in \{0,1\}^{p(n)}$ say that r is *bad for* x iff $M(x,r) \neq A(x)$. Therefore, for all $x \in \{0,1\}^n$,

$$\Pr_r[r \text{ is bad for } x] \leq 2^{-2n}$$

We say that r is *bad* if there exists an $x \in \{0,1\}^n$ such that r is bad for x.

$$\begin{aligned}
\Pr[r \text{ is bad}] &= \Pr[\exists x \in \{0,1\}^n.\ r \text{ is bad for } x] \\
&\leq \sum_{x \in \{0,1\}^n} \Pr[r \text{ is bad for } x] \\
&\leq 2^n 2^{-2n} \\
&< 1.
\end{aligned}$$

Thus $\Pr[r \text{ is not bad}] > 0$. Therefore there must exist an $r_n \in \{0,1\}^{p(n)}$ such that r_n is not bad. Apply this same argument for each value of n and use this sequence $\{r_n\}_{n=0}^{\infty}$ as the advice sequence to a $\mathsf{P/poly}$ machine that decides A. Each advice string is a particular random string that leads to a correct answer for every input of that length. □

3.6 BPP and the Polynomial Time Hierarchy

Here we show that randomness ican be simulated by a small amount of alternation.

Theorem 3.5 (Sipser-Gacs, Lautemann). $\mathsf{BPP} \subseteq \Sigma_2\mathsf{P} \cap \Pi_2\mathsf{P}$.

Proof. Note that BPP is closed under complement; so, it suffices to show $\mathsf{BPP} \subseteq \Sigma_2 P$.
Let $A \in \mathsf{BPP}$. Then there is a probabilistic polytime TM M and polynomial p such that for $x \in \{0,1\}^n$,

$$\Pr_{r \in \{0,1\}^{p(n)}}[M(x,r) \neq A(x)] \leq 2^{-n}.$$

Define $Acc_M(x) = \{r \in \{0,1\}^{p(n)} \mid M(x,r) = 1\}$. In order to determine whether or not $x \in A$ we need to determine whether the set $S = Acc_M(x)$ is large (nearly all of $\{0,1\}^{p(n)}$) or small (only an exponentially small fraction of $\{0,1\}^{p(n)}$).

The basic idea of the method we use is the following: If a set S contains a large fraction of $\{0,1\}^{p(n}$ then a small number of "translations" of S will cover $\{0,1\}^{p(n)}$. If S is a very small fraction of $\{0,1\}^{p(n)}$ then no small set of "translations" of S will suffice to cover $\{0,1\}^{p(n)}$.

The translation we will use is just bitwise exclusive or, $\oplus$. We will use the following invertibility property of $\oplus$: for $s \in \{0,1\}^m$, $t \in \{0,1\}^m$,

$$b = s \oplus t \Longleftrightarrow t = s \oplus b$$

For $S \subseteq \{0,1\}^m$, define $S \oplus t = \{s \oplus t | s \in S\}$. Note that $|S \oplus t| = |S|$.

Lemma 3.6 (Lautemann's Lemma). *Let $S \subseteq \{0,1\}^m$. If $|S|/2^m \geq 1/2$ then there exist $t_1 \ldots t_m \in \{0,1\}^m$ such that, $\bigcup_{j=1}^m (S \oplus t_j) = \{0,1\}^m$.*

Thus the number of translations required is only linear in the number of bits in the size of the universe from which S is chosen.

Proof. By probabilistic method. Let S satisfy the conditions of the lemma.

Fix a string $b \in \{0,1\}^m$. Choose $t_1 \ldots t_m$ uniformly and independently at random from $\{0,1\}^m$. For any $j \in \{1, \ldots, m\}$,

$$\begin{aligned} \Pr[b \in S \oplus t_j] &= \Pr[t_j \in S \oplus b] \\ &= \Pr[t_j \in S] \qquad \text{since } |S \oplus t_j| = |S| \\ &\geq 1/2. \end{aligned}$$

Therefore $\Pr[b \notin S \oplus t_j] < 1/2$. Thus, by independence, the probability that b is not in any of the m translations

$$\Pr[b \notin \bigcup_{j=1}^{m} (S \oplus t_j)] < 2^{-m}.$$

Thus

$$\Pr[\exists b \in \{0,1\}^m.\ b \notin \bigcup_{j=1}^{m}(S \oplus t_j)] < 2^m 2^{-m} = 1,$$

and so

$$\Pr[\forall b \in \{0,1\}^m.\ b \in \bigcup_{j=1}^{m}(S \oplus t_j)] > 0.$$

Therefore there exists a set $t_1 \ldots t_m$ such that the union of the translations of S by t_i covers all strings in $\{0,1\}^m$. □

Now apply Lautemann's Lemma with $S = Acc_M(x)$ and $m = p(n)$. If $x \notin A$ then only a 2^{-n} fraction of the random strings will be in $Acc_M(x)$, and so $m = p(n)$ translations will not be able to cover all of $\{0,1\}^{p(n)}$. This gives us the following Σ_2P characterization of A:

$x \in A \iff \exists t_1 \ldots t_{p(|x|)} \in \{0,1\}^{p(|x|)} \forall^{p(|x|)} r$ there is some $j \in \{1, \ldots, p(|x|)\}$ such that $M(x, r \oplus t_j) = 1$.

Note that there are only polynomial many j values to be checked, so these can be checked directly by the machine in polynomial time. □

Lecture 4

Circuit Complexity and the Polytime Hierarchy

April 8, 2004
Lecturer: Paul Beame
Notes: Ashish Sabharwal

So far we have seen that circuits are quite powerful. In particular, $\mathsf{P/poly}$ contains undecidable problems, and $\mathsf{RP} \subseteq \mathsf{BPP} \subseteq \mathsf{P/poly}$. In this lecture, we will explore this relationship further, proving results that show circuits are very unlikely to be super-powerful compared to uniform complexity classes.

Theorem 4.1. *A.* **(Shannon, 1949)** *"Most" Boolean functions $f : \{0,1\}^n \to \{0,1\}$, have circuit complexity $\mathsf{SIZE}(f) \geq \frac{2^n}{n} - \phi_n$, where ϕ_n is $o\left(\frac{2^n}{n}\right)$. (More precisely, for any $\epsilon > 0$ this holds for at least a $(1-\epsilon)$ fraction of all Boolean functions.)*

B. **(Lupanov, 1965)** *Every Boolean function $f : \{0,1\}^n \to \{0,1\}$ can be computed in $\mathsf{SIZE}(f) \leq \frac{2^n}{n} + \theta_n$, where θ_n is $o\left(\frac{2^n}{n}\right)$.*

Proof. A. The proof is a by a counting argument. Let $\mathbb{B}_n = \{f : \{0,1\}^n \to \{0,1\}\}$, that is, the set of all Boolean functions on n bits. $|\mathbb{B}_n| = 2^{2^n}$. We will show that the number of circuits of size much smaller than $2^n/n$ is only a negligible fraction of $|\mathbb{B}_n|$, proving the claim.

Let us compute the number of circuits of size at most $S \geq n+2$ over $\{\neg, \wedge, \vee\}$. Note that the argument we present works essentially unchanged for any complete basis of gates for Boolean circuits. What does it take to specify a given circuit? A gate labeled i in the circuit is defined by the labels of its two inputs, j and k ($j = k$ for unary gates), and the operation g the gate performs. The input labels j and k can be any of the S gates or the n inputs or the two constants, 0 and 1. The operation g can be any one of the three Boolean operations in the basis $\{\neg, \wedge, \vee\}$. Adding to this the name i of the gate, any circuit of size at most S can be specified by a description of length at most $(S+n+2)^{2S}3^S S$. Note, however, that such descriptions are the same up to the $S!$ ways of naming the gates. Hence, the total number of gates of size at most S, noting that $S! \geq (S/e)^S$, is at most

$$\begin{aligned}
\frac{(S+n+2)^{2S}3^S S}{S!} &\leq \frac{(S+n+2)^{2S}(3e)^S S}{S^S} \\
&= \left(\frac{S+n+2}{S}\right)^S (3e\,(S+n+2))^S\, S \\
&= \left(1+\frac{n+2}{S}\right)^S (3e\,(S+n+2))^S\, S \\
&\leq \left(e^{\frac{n+2}{S}}\, 3e\,(S+n+2)\right)^S S \qquad \text{since } 1+x \leq e^x \\
&< (6e^2 S)^{S+1} \qquad \text{since we assumed } S \geq n+2.
\end{aligned}$$

To be able to compute at least an ϵ fraction of all functions in $\mathbb{B}_n$, we need

$$\begin{array}{rrcl} & (6e^2S)^{S+1} & \geq & \epsilon\, 2^{2^n} \\ \Rightarrow & (S+1)\log_2(6e^2S) & \geq & 2^n - \log_2(1/\epsilon) \\ \Rightarrow & (S+1)(5.5+\log_2 S) & \geq & 2^n - \log_2(1/\epsilon) \end{array}$$

Hence, we must have $S \geq 2^n/n - \phi_n$ where ϕ_n is $o(2^n/n)$ to compute at least an ϵ fraction of all functions in $\mathbb{B}_n$ as long as ϵ is $2^{-o(2^n)}$. This proves part A of the Theorem.

B. Proof of this part is left as an exercise (see Problem 3, Assignment 1). Note that a Boolean function over n variables can be easily computed in $\mathsf{SIZE}(n2^n)$ by using its canonical DNF or CNF representation. Bringing it down close to $\mathsf{SIZE}(2^n/n)$ is a bit trickier.

□

This gives a fairly tight bound on the size needed to compute most Boolean functions over n variables. As a corollary, we get a circuit size hierarchy theorem which is even stronger than the time and space hierarchies we saw earlier; circuits can compute many more functions even when their size is only roughly doubled.

Corollary 4.2 (Circuit-size Hierarchy). *For any $\epsilon > 0$ and $S_1, S_2 : \mathbb{N} \to \mathbb{N}$, if $n \leq (2+\epsilon)S_1(n) \leq S_2(n) \ll 2^n/n$, then $\mathsf{SIZE}(S_1(n)) \subsetneq \mathsf{SIZE}(S_2(n))$.*

Proof. Let $m = m(n)$ be the maximum integer such that $S_2(n) \geq (1+\epsilon/2)\, 2^m/m$. By the preconditions of the Corollary, $S_1(n) \leq (1-\epsilon/2)\, 2^m/m$ and $m \ll n$. Consider the set $\mathcal{F}$ of all Boolean functions on n variables that depend only on m bits of their inputs. By the previous Theorem, all functions in $\mathcal{F}$ can be computed by circuits of size $2^m/m + o(2^m/m)$ and are therefore in $\mathsf{SIZE}(S_2(n))$. On the other hand, most of the functions in $\mathcal{F}$ cannot be computed by circuits of size $2^m/m - o(2^m/m)$ and are therefore not in $\mathsf{SIZE}(S_1(n))$. □

The following theorem, whose proof we will postpone until the next lecture, shows that circuits can quite efficiently simulate uniform computation. Its corollaries will be useful in several contexts.

Theorem 4.3 (Pippenger-Fischer, 1979). *If $T(n) \geq n$, then $\mathsf{TIME}(T(n)) \subseteq \bigcup_c \mathsf{SIZE}(cT(n)\log_2 T(n))$.*

We now show that although $\mathsf{P/poly}$ contains undecidable problems, it is unlikely to contain even all of NP. This implies that circuits, despite having the advantage of being non-uniform, may not be all that powerful. We start with a simple exercise:

Theorem 4.4 (Karp-Lipton). *If $\mathsf{NP} \subseteq \mathsf{P/poly}$, then $\mathsf{PH} = \Sigma_2\mathsf{P} \cap \Pi_2\mathsf{P}$.*

The original paper by Karp and Lipton credits Sipser with sharpening the result. The proof below which uses the same general ideas in a slightly different way is due to Wilson.

Proof. Suppose to the contrary that $\mathsf{NP} \subseteq \mathsf{P/poly}$. We'll show that this implies $\Sigma_2\mathsf{P} = \Pi_2\mathsf{P}$. From Lemma 2.6 this will prove the Theorem.

Let $L \in \Pi_2\mathsf{P}$. Therefore there exists a polynomial-time computable set R and a polynomial p such that $L = \left\{x \mid \forall^{p(|x|)} y\, \exists^{p(|x|)} z.\ (x,y,z) \in R\right\}$. The idea behind the proof is as follows. The inner relation in this definition, $\left\{(x,y) \mid \exists^{p(|x|)} z.\ (x,y,z) \in R\right\}$, is an NP language. $\mathsf{NP} \subseteq \mathsf{P/poly}$ implies that there exists a

polynomial size circuit family $\{C_R\}$ computing this inner relation. We would like to simplify the definition of L using this circuit family. by

$$\left\{ x \mid \exists \langle C_R \rangle \; \forall^{p(|x|)} y.\; C_R \text{ correctly computes } R \text{ on } (x,y) \text{ and } C_R(x,y) = 1 \right\}.$$

This would put L in $\Sigma_2\mathsf{P}$, except that it is unclear how to efficiently verify that C_R actually computes the correct inner relation corresponding to R. (Moreover, the whole circuit family may not have a finite specification.)

To handle this issue, we modify the approach and use *self-reduction* for NP to verify correctness of the circuit involved. More precisely, we create a modified version of R suitable for self-reduction. Let

$$R' = \left\{ (x,y,z') \mid |z'|, |y| \leq p(|x|) \text{ and } \exists^{p(|x|)-|z'|} z''.\; (x,y,z',z'') \in R \right\}.$$

Here z' acts as a prefix of z in the earlier definition of R. Note that $R' \in \mathsf{NP}$ since R is polynomial-time computable. Therefore, by the assumption $\mathsf{NP} \subseteq \mathsf{P/poly}$, R' is computed by a polynomial size circuit family $\{C_n\}_{n=0}^{\infty}$ with a polynomial size bound $q : \mathbb{N} \to \mathbb{N}$. We, of course, can't encode the whole circuit family for showing $L \in \Sigma_2\mathsf{P}$. We use the fact that on input x, we only query R' on inputs (x,y,z) of length at most $2(|x| + 2p(|x|))$, say, assuming some reasonable encoding of the tuples.

Let $\mathcal{C}_{pref,|x|}$ be the smallest prefix of $\{C_n\}_n$ that contains circuits corresponding to all input sizes that are queried. The size of this is bounded some polynomial q' that involves the composition of p and q. We claim that there exists a polynomial-time algorithm M that given x, y and $\mathcal{C}_{pref,|x|}$ as input, either

a. outputs a z such that $(x,y,z) \in R$, in which case there exists a z satisfying this property, or

b. fails, in which case either $\mathcal{C}_{pref,|x|}$ is not a prefix of $\{C_n\}_{n=0}^{\infty}$ for computing the NP set R', or no such z exists.

We prove the claim by describing an algorithm M that behaves as desired. It will be clear that M runs in polynomial time.

Algorithm M: On input $x, y, \mathcal{C}_{pref,|x|}$,
  Let z' be the empty string
  If $\mathcal{C}_{pref,|x|}(x,y,z') = 0$ then fail
  While $(x,y,z') \notin R$ and $|z'| \leq p(|x|)$
    If $\mathcal{C}_{pref,|x|}(x,y,z'0) = 1$
      then $z' \leftarrow z'0$
      else $z' \leftarrow z'1$
    EndIf
  EndWhile
  If $(x,y,z') \in R$
    then output z'
    else fail
  EndIf
End

Given M satisfying the conditions of our claim above, we can characterize the language L as follows: $x \in L$ iff $\exists^{q'(|x|)} \langle \mathcal{C}_{pref,|x|} \rangle \; \forall^{p(|x|)} y.\; M^{decision}(x, y, \langle \mathcal{C}_{pref,|x|} \rangle)$. Here $M^{decision}$ denotes the decision version of M that outputs true or false rather than z' or fail. Since M is polynomial-time computable, this shows that $L \in \Sigma_2\mathsf{P}$. Note that we were able to switch $\exists$ and $\forall$ quantifiers because $\mathcal{C}_{pref,|x|}$ doesn't depend on y.

This proves that $\Pi_2\mathsf{P} \subseteq \Sigma_2\mathsf{P}$. By the symmetry between $\Sigma_2\mathsf{P}$ and $\Pi_2\mathsf{P}$, this implies $\Sigma_2\mathsf{P} \subseteq \Pi_2\mathsf{P}$, making the two classes identical and finishing the proof. □

The following exercise uses the same kind of self reduction that we employed in the above argument:

Exercise 4.1. Prove that $\mathsf{NP} \subseteq \mathsf{BPP}$ implies $\mathsf{NP} = \mathsf{RP}$.

We now prove that even very low levels of the polynomial time hierarchy cannot be computed by circuits of size n^k for any fixed k. This result, unlike our previous Theorem, is *unconditional*; it does not depend upon our belief that the polynomial hierarchy is unlikely to collapse.

Theorem 4.5 (Kannan). *For all k, $\Sigma_2\mathsf{P} \cap \Pi_2\mathsf{P} \not\subseteq \mathsf{SIZE}(n^k)$.*

Proof. We know that $\mathsf{SIZE}(n^k) \subsetneq \mathsf{SIZE}(n^{k+1})$ by the circuit hierarchy theorem. To prove this Theorem, we will give a problem in $\mathsf{SIZE}(n^{k+1})$ and $\Sigma_2\mathsf{P} \cap \Pi_2\mathsf{P}$ that is not in $\mathsf{SIZE}(n^k)$.

For each n, let C_n be the lexically first circuit on n inputs such that $size(C_n) \geq n^{k+1}$ and C_n is minimal; *i.e.*, C_n is not equivalent to a smaller circuit. (For lexical ordering on circuit encodings, we'll use $\prec$.) Let $\{C_n\}_{n=0}^{\infty}$ be the corresponding circuit family and let A be the language decided by this family. By our choice of C_n, $A \notin \mathsf{SIZE}(n^k)$. Also, by the circuit hierarchy theorem, $size(A)$ is a polynomial $\leq (2+\epsilon)n^{k+1}$ and the size of its encoding $|\langle A \rangle| \leq n^{k+3}$, say. Note that the factor of $(2+\epsilon)$ is present because there may not be a circuit of size exactly n^{k+1} that computes A, but there must be one of size at most roughly twice this much.

Claim: $A \in \Sigma_4\mathsf{P}$.
The proof of this claim involves characterizing the set S using a small number of quantifiers. By definition, $x \in A$ if and only if

$$\begin{aligned}\exists^{p(|x|)}\langle C_{|x|}\rangle. \quad & \big(size(C_{|x|}) \geq |x|^{k+1} \\ & \bigwedge \forall^{p(|x|)}\langle D_{|x|}\rangle.\ [size(D_{|x|}) < size(C_{|x|}) \rightarrow \exists^{|x|} y.\ D_{|x|}(y) \neq C_{|x|}(y)] \\ & \bigwedge \forall^{p(|x|)}\langle D_{|x|}\rangle.\ [\ [(\langle D_{|x|}\rangle \prec \langle C_{|x|}\rangle) \wedge (size(D_{|x|}) \geq |x|^{k+1})] \rightarrow \\ & \qquad \exists^{p(|x|)}\langle E_{|x|}\rangle.\ [size(E_{|x|}) < size(D_{|x|}) \wedge \forall^{|x|} z.\ D_{|x|}(z) = E_{|x|}(z)]\ ]\big)\end{aligned}$$

The second condition states that the circuit is minimal, *i.e.*, no smaller circuit $D_{|x|}$ computes the same function as $C_{|x|}$. The third condition enforces the lexically-first requirement; *i.e.*, if there is a lexically-earlier circuit $D_{|x|}$ of size at least $|x|^{k+1}$, then $D_{|x|}$ itself is not minimal as evidenced by a smaller circuit $E_{|x|}$. When we convert this formula into prenex form, all quantifiers, being in positive form, do not flip. This gives us that $x \in A$ iff $\underbrace{\exists\langle C_{|x|}\rangle}\ \underbrace{\forall\langle D_{|x|}\rangle}\ \underbrace{\exists^{|x|} y\ \exists\langle E_{|x|}\rangle}\ \underbrace{\forall^{|x|} z}.\ \phi$ for a certain quantifier free polynomially decidable formula ϕ. Hence $A \in \Sigma_4\mathsf{P}$.

This proves the claim and imples that $\Sigma_4\mathsf{P} \not\subseteq \mathsf{SIZE}(n^k)$. We finish the proof of the Theorem by analyzing two possible scenarios:

a. $\mathsf{NP} \subseteq \mathsf{P/poly}$. In this case, by the Karp-Lipton Theorem, $A \in \Sigma_4\mathsf{P} \subseteq \mathsf{PH} = \Sigma_2\mathsf{P} \cap \Pi_2\mathsf{P}$ because the polynomial time hierarchy collapses, and we are done.

b. $NP \not\subseteq \mathsf{P/poly}$. In this simpler case, for some $B \in \mathsf{NP}$, $B \notin \mathsf{P/poly}$. This implies $B \notin \mathsf{SIZE}(n^k)$ and proves, in particular, that $\Sigma_2\mathsf{P} \cap \Pi_2\mathsf{P} \not\subseteq \mathsf{SIZE}(n^k)$.

This finishes the proof of the Thoerem. We note that unlike the existential argument (the witness is either the language A or the language B), one can also define a single language A' witnessing it where A' is a hybrid language between A and a diagonal language in NP. □

Lecture 5

Circuit Complexity / Counting Problems

April 12, 2004
Lecturer: Paul Beame
Notes: William Pentney

5.1 Circuit Complexity and Uniform Complexity

We will conclude our look at the basic relationship between circuit complexity classes and uniform complexity classes. First, we will prove that circuits can efficiently solve problems in uniform complexity classes.

Theorem 5.1 (Pippenger, Fischer). *If $T(n) \geq n$, then $\mathsf{TIME}(T(n)) \subseteq \bigcup \mathsf{SIZE}(kT(n)\log_2 T(n))$.*

First, consider the following variant of the traditional k-tape TM:

Definition 5.1. A multitape TM is *oblivious* if the motion of each head depends only upon the length of its input.

Note that an oblivious TM's motion has nothing to do with the input itself; given an input of size n, the machine's head(s) will perform the *same* series of motions regardless of the original contents of the tape. We will prove two lemmas from which Theorem 5.1 follows immediately.

The first is a more careful version of the original result of Hennie and Stearns showing that k-tape TMs can be efficiently simulated by 2-tape TMs. The key extension is that the resulting TM can be made oblivious.

Lemma 5.2 (Hennie-Stearns, Pippenger-Fischer). *For $T(n) \geq n$, if $A \in \mathsf{TIME}(T(n))$ then A is recognized by a 2-tape deterministic oblivious TM in time $O(T(n)\log T(n))$.*

The second lemma shows that oblivious TMs can be efficiently simulated by circuits.

Lemma 5.3. *[Pippenger-Fischer] If $A \subseteq \{0,1\}^*$ is recognized by a 2-tape deterministic oblivious TM in time $O(T(n))$, then $A \in \bigcup_k \mathsf{SIZE}(kT(n))$ (i.e. size linear in $T(n)$.*

First we will prove Lemma 5.3 whose proof motivates the notion of oblivious TMs.

Proof of Lemma 5.3. Consider the tableau generated in the standard proof of the Cook-Levin theorem for an input of size n for a Turing machine running in $O(T(n)$. The tableau yields a circuit of size $O(T^2(n))$, with "windows" of size 2×3. However, on any given input the TM head will only be in one cell at a given time step; i.e., one cell per row in the tableau. With a normal TM, though, it is not possible to anticipate which cell that will be.

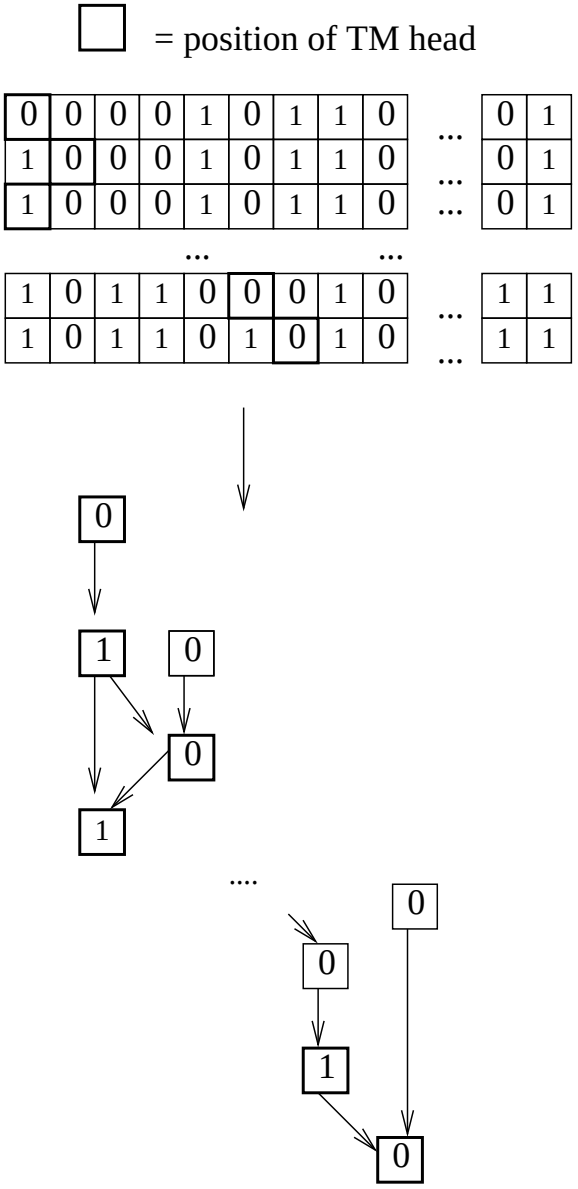

Figure 5.1: Representing an oblivious TM using constant-size windows

.

Since our TM M is oblivious, we know the path that the head shall take, and where it will be at each timestep, simply by knowing the length of the input. We know if and when a given cell on the tape will be revisited, and can ignore it at other times. Thus we can use "windows" containing simply the contents of a cell from the last time it was active and the contents of the previous cell visited to see what the head moves, to represent each step taken by that head, and all other cells in the tableau will be irrelevant to that step in the computation. This is represented in Figure 5.1. The subsequent configuration resulting from any configuration can thus be computed by a circuit of constant size. Since the tableau is of height $T(n)$, we can express the tableau in size $O(T(n))$. □

Now, we prove Lemma 5.2, that if $A \in \mathsf{TIME}(T(n))$, then A is recognized by a 2-tape deterministic

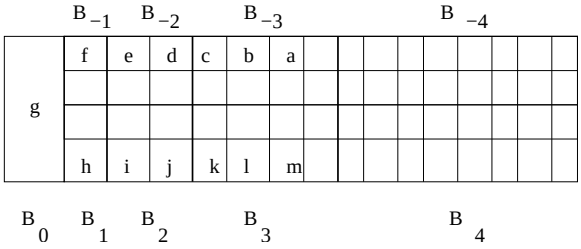

Figure 5.2: 4-track oblivious simulation of a TM tape.

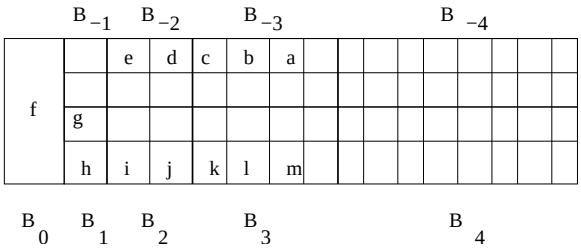

Figure 5.3: The simulation in Figure 5.2 when the head moves left. Block B_1 is bunched and B_{-1} stretched.

and oblivious TM in time $O(T(n) \log T(n))$.

Proof of Lemma 5.2. We will build a 2-tape TM M' which will consist of two tapes: one holding all the contents of M's tapes, and one "work tape".

Traditionally, we think of a TM as having a tape which the head moves back on forth on. For this proof, it may help to instead think of the TM head as being in a fixed position and each of the tapes as moving instead, similar to a typewriter carriage (if anyone remembers this ancient beasts!). Of course a tape would be an infinitely long typewriter carriage which would be too much to move in a single step. Instead, we think of moving the tapes back and forth, but we may not do so smoothly; sometimes we will be "bunching up" a tape, such that some cells may be effectively layered on top of others and other parts will be "stretched", parts that are not bunched or stretched are "smooth".

It will be convenient to concentrate on simulating one tape of M at a time. In reality we will allocate several tracks on tape 1 of M' to each of the k-tapes. It is simplest also to imagine that each of M's tapes is 2-way infinite, although M' will have only 1-way infinite tapes.

More precisely, imagine the scenario modeled in Figure 5.2. For each tape of M, we have four tracks – two for those cells to the left of the current cell, and two for those to the right of the current cell. The "outer" tracks will contain the contents of the tape when that portion of the tape is smooth; the "inner" tracks will contain portions of the tape that are bunched up. The current cell is always on the left end of the set of tracks. The tracks wrap around as the head of M moves.

We will divide the tracks into "blocks", where each block will consist of a portion of the two upper or lower tracks. Block B_0 contains the current cell (at the left end of the tape); block B_1 is the lower half of the the cell immediately to the right of block B_0 and has capacity for up to 2 cells of M; block B_2 is the lower half of the next 2 cells to the right of that with capacity for up to 4 cells of M, etc. In general, block B_i has contains the lower half of the next 2^{i-1} cells starting at cell 2^{i-1} and has capacity for 2^i cells of M. Block B_{-i} for $i \geq 1$ contains the corresponding upper half of cells of M' whose lower half is the block B_i. In general, for block B_i and B_j, if $i < j$ then the contents of block B_i will be to the left of those for block

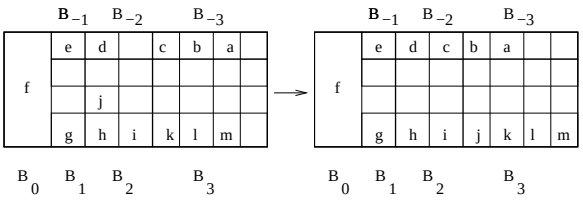

Figure 5.4: The smoothing of blocks B_2 and block B_{-2}.

B_j. The outer tracks of block B_i or B_{-i} will contain contents of cells of M that are closer to the cell in block B_0 than the inner tracks.

In our simulation, when we wrap the tracks around, we permit the tape to "bunch up" within a block; when this happens, the contents of the cell of M that would ordinarily be in the outer track of the next block is now placed in the next free cell on the inner track of the current block. We can later undo this by moving the bunched-up data in block B_i into B_{i+1}; this process is called *smoothing* the block. Similarly, a block B_i may be stretched, so that it contains fewer than 2^{i-1} cells from the tape; if other data is pushed into it, the vacant area on the track is filled. The tape is smoothed when it is no longer stretched. Figure 5.3 shows the movement of the tracks in our simulation depicted in Figure 5.2 when the head moves left; block B_1 is bunched and block B_{-1} stretched. In Figure 5.4, we see the smoothing of blocks B_2 and B_{-2} on such a simulation.

We maintain the following invariants after simulating each time step t of the simulation:

A. Every block B_i with $|i| > \lceil \log_2 t \rceil + 1$ is smooth.

B. $B_i + B_{-i} = 2^{|i|}$ i.e. if B_i is bunched, B_{-i} is stretched an equal amount.

C. Each B_i consists of consecutive elements of the tape of M.

D. If t is a multiple of 2^i then blocks $B_i, B_{i-1}, B_{i-2}..., B_1, B_0, B_{-1}, B_{-2}, ...B_{-i}$ are smooth.

To maintain our invariants, whenever t is $2^i t'$ for some odd t', we smooth out blocks $B_{-i}...B_i$ by borrowing from or adding entries to B_{i+1} and $B_{-(i+1)}$. We do this by copying the contents of all the cells in $B_1, \ldots, B_i, B_{i+1}$ onto tape 2 and then writing them back into those cells only on the outer tracks, except, possibly, for block B_{i+1}. The same is done for blocks $B_{-1}, \ldots, B_{-(i+1)}$. We need to argue that this smoothing can be accomplished.

After smoothing blocks $B_{-(i+1)}, \ldots, B_{(i+1)}$ when t is a multiple of 2^{i+1}, we may simulate M for up $2^i - 1$ steps without accessing blocks B_{i+1} or $B_{-(i+1)}$. On the 2^i-th step the number of entries that will need to appear in blocks $B_0, B_1, \ldots, B_i$ (or $B_{-i}, B_{i-1}, \ldots, B_0$) is between 0 and 2^{i+1} since this number can have changed by at most 2^i during this time. Since B_{i+1} has remained smooth during this time, there are precisely 2^i occupied cells in B_{i+1} and space for 2^i more inputs. Therefore there enough entries in B_{i+1} to borrow to fill the up to 2^i spaces in blocks $B_0, \ldots, B_i$ that need to be filled, or enough space to take the overflow from those blocks. A similar argument applies to blocks $B_{-i}, \ldots, B_0$. Therefore this smoothing can be accomplished. By carefully making sure that the inner and outer tracks are scanned whether or not they contain entries, the smoothing can be done with oblivious motion.

How much does the smoothing cost when t is a multiple of 2^i but not 2^{i+1}? It will be less than or equal to $c2^i$ steps in tota for some constant c. We will perform smoothing $\frac{1}{2^i}$ of the time, with cost $c2^i$ per smoothing. Therefore, the total time will be $\sum_{i \leq \lceil \log_2 T(n) \rceil + 1} c2^i \frac{T(n)}{2^i}$ which is $O(T(n) \log T(n)$. □

We can use the proof of THeoremm 5.1 to prove the following theorem:

Theorem 5.4 (Cook 83). *If $T(n) \leq n$ and $L \in \mathsf{NTIME}(T(n))$ then there exists a reduction that maps input x to a 3-CNF formula ϕ_x in $O(T(n)\log T(n))$ time and $O(\log T(n))$ space such that ϕ_x has size $O(T(n)\log T(n))$ and $x \in L \Leftrightarrow \phi_x \in 3SAT$.*

Proof. ϕ_x is usual 3-CNF based on CIRCUIT-SAT. Check that producing the circuit in the proof of the previous lemmas can be done in $O(T(n)\log T(n))$ time and $O(\log T(n))$ space. □

5.2 Function Complexity Classes, Counting Problems, and $\#\mathsf{P}$

We will now define some complexity classes of numerical functions on input strings rather than decision problems.

Definition 5.2. The class FP is the set of all functions $f : \{0,1\}^* \to \mathbb{N}$ that are computable in polynomial time.

Among the algorithmic problems representable by such functions are "counting problems" pertaining to common decision problems. For example,

Definition 5.3. Define #3-SAT as the problem of counting the number of satisfying assignments to a given 3-SAT formula ϕ.

Remark. Note that if we can solve #3-SAT in polynomial time, then clearly we can solve any NP-complete problem in polynomial time and thus $\mathsf{P} = \mathsf{NP}$.

We can define new complexity classes to represent such counting problems:

Definition 5.4. For any complexity class C, let $\#\mathsf{C}$ be the set of all functions $\{f : \{0,1\}^* \to \mathbb{N}$ for which there exists $R \in \mathsf{C}$ and a polynomial $p : \mathbb{N} \to \mathbb{N}$ such that $f(x) = \#\{y \in \{0,1\}^{p(|x|)} : (x,y) \in R\} = |\{y \in \{0,1\}^{p(|x|)} : (x,y) \in R\}|$.

In particular, for $\mathsf{C} = \mathsf{P}$, we obtain

Definition 5.5. Define the class $\#\mathsf{P}$ to be the set of all functions $\{f : \{0,1\}^* \to \mathbf{N}$ for which there exists $R \in \mathsf{P}$ and a polynomial $p : \mathbb{N} \to \mathbb{N}$ such that $f(x) = \#\{y \in \{0,1\}^{p(|x|)} : (x,y) \in R\} = |\{y \in \{0,1\}^{p(|x|)} : (x,y) \in R\}|$.

5.2.1 Function classes and oracles

Let us consider the use of oracle TMs in the context of function classes. For a language A, let FP^A be the set of functions $f : \{0,1\}^* \to \mathbb{N}$ that can be solved in polynomial time by an oracle TM $M^?$ that has A as an oracle.

Similarly, we can define oracle TM's $M^?$ that allow functions as oracles rather than sets. In this case, rather than receiving the answer from the oracle by entering one of two states, the machine can receive a binary encoded version of the oracle answer on an oracle answer tape. Thus for functions $f : \{0,1\}^* \to \mathbb{N}$ and a complexity class C for which it makes sense to define oracle versions, we can define C^f, and for a complexity class FC' of functions we can define $\mathsf{C}^{\mathsf{FC}'} = \bigcup_{f \in \mathsf{FC}'} \mathsf{C}^f$.

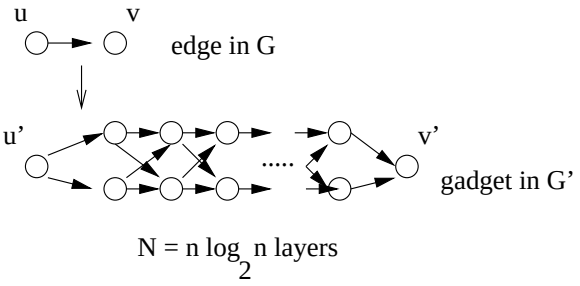

Figure 5.5: Gadgets used in reduction of #HAM-CYCLE to #CYCLE.

Definition 5.6. A function f is $\#\mathsf{P}$*-complete* iff

A. $f \in \#\mathsf{P}$.

B. For all $g \in \#\mathsf{P}, g \in \mathsf{FP}^f$.

As 3-SAT is NP-complete, #3-SAT is $\#\mathsf{P}$-complete:

Theorem 5.5. *#3-SAT is* $\#\mathsf{P}$*-complete.*

Proof. Cook's reduction is "parsimonious", in that it preserves the number of solutions. More precisely, in circuit form there is precisely one satisfying assignment for the circuit for each NP witness y. Moreover, the conversion of the circuit to 3-SAT enforces precisely one satisfying assignment for each of the extension variables associated with each gate. □

Since the standard reductions are frequently parsimonious, and can be used to prove $\#\mathsf{P}$-completeness of many counting problems relating to NP-complete problems. In some instances they are not parsimonious but can be made parsimonious. For example we have the following.

Theorem 5.6. #HAM-CYCLE *is* $\#\mathsf{P}$*-complete.*

The set of $\#\mathsf{P}$-complete problems is not restricted to the counting versions of NP-complete problems, however; interestingly, problems in P can have $\#\mathsf{P}$-complete counting problems as well. Consider #CYCLE, the problem of finding the number of directed simple cycles in a graph G. (The corresponding problem CYCLE is in P).

Theorem 5.7. *#CYCLE is* $\#\mathsf{P}$*-complete.*

Proof. We reduce from #HAM-CYCLE. We will map the input graph G for #HAM-CYCLE to a graph G' for #CYCLE. Say G has n vertices. G' will have a copy u' of each vertex $u \in G$, and for each edge $(u, v) \in G$ the gadget in Figure 5.5 will be added between u' and v' in G'. This gadget consists of $N = n\lceil \log_2 n \rceil + 1$ layers of pairs of vertices, connected to u and v and connected by $4N$ edges within. The number of paths from u' to v' in G' is $2^N > n^n$. Each simple cycle of length ℓ in G yields $2^{N\ell}$ simple cycles in G'. If G has k Hamiltonian cycles, there will be $k2^{Nn}$ corresponding simple cycles in G'. G has at most n^n simple cycles of length $\leq n-1$. The total number of simple cycles in G' corresponding to these is $\leq n^n 2^{N(n-1)} < 2^{Nn}$. Therefore we compute #HAM-CYCLE$(G) = \lfloor$#CYCLE$(G')/2^{Nn}\rceil$. □

The following theorem is left as an exercise:

Theorem 5.8. *#2-SAT is* $\#\mathsf{P}$*-complete.*

5.2.2 Determinant and Permanent

Some interesting problems in matrix algebra are represented in function complexity classes. Given an $n \times n$ matrix A, the determinant of A is

$$det(A) = \sum_{\sigma \in S_n} (-1)^{sgn(\sigma)} \prod_{i=1}^{n} a_{i,\sigma(i)},$$

where S_n is the set of permutations of n and $sgn(\sigma)$ is the number of trnaspositions required to produce σ modulo 2. This problem is in FP.

The $sgn(\sigma)$ is apparently a complicating factor in the definition of $det(A)$, but if we remove it we will see that the problem actually becomes harder. Given an $n \times n$ matrix A, the *permanent* of A is equal to

$$perm(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} a_{i,\sigma(i)}.$$

Let 0-1PERM be the problem of finding the permanent of a 0-1 matrix. When we continue, we will prove the following theorem:

Theorem 5.9 (Valiant). *0-1PERM is #P-complete.*

The following is an interesting interpretation of the permanent. We can view the matrix A the adjacency matrix of a weighted bipartite graph on vertices $[n] \times [n]$ where $[n] = \{1, \ldots, n\}$. Each $\sigma \in S_n$ corresponds to a perfect matching of this graph. If we view the weight of a matching as the product of the weights of its edges the permanent is the total weight of all matchings in the graph.

In particular a 0-1 matrix A corresponds to an unweighted bipartite graph G for which A is the adjacency matrix, and Perm(A) represents the total weight of all perfect matchings on G. Let #BIPARTITE-MATCHING be the problem of counting all such matchings. Thus we obtain the following corollary as well:

Corollary 5.10. *#BIPARTITE-MATCHINGS is $\#$P-complete*

Lecture 6

#P, the Permanent, and Toda's Theorem

April 15, 2004
Lecturer: Paul Beame
Notes: Nilesh Dalvi

In this lecture, we will prove that the problem of finding the permanent of a 0-1 matrix is #P-complete. Given an $n \times n$, 0-1 matrix A, it can be viewed as an adjacency matrix of a directed graph G on n vertices (with possibly self-loops). It is easy to see that the permanent of A is the number of cycle-covers of G. (A cycle-cover is a sub-graph consisting of a union of disjoint cycles that cover the vertices of G).

The hardness of 0-1PERM is established by showing that the problem of finding the number of cycle-covers of G is hard.

Theorem 6.1 (Valiant). 0-1PERM *is #P-complete.*

Proof. For a directed graph G, a *cycle-cover of* G is a union of simple cycles of G that contains each vertex precisely once. For a weighted, directed graph G, with weight matrix A, we can also view

$$\text{PERM}(G) = \text{PERM}(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} a_{i,\sigma(i)}$$

as the total weight of all cycle-covers of G, where the weight of a cycle-cover is the product of the weights of all its edges. This interpretation corresponds naturally to the representation of a permutation σ as a union of directed cycles. For example, if $\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 3 & 4 & 5 & 2 & 1 & 6 \end{pmatrix} \in S_6$ then σ can also be written in cycle form as $(1\ 3\ 5)(2\ 4)(6)$ where the notation implies that each number in the group maps to the next and the last maps to the first. (See Figure 6.1.) Thus, for an unweighted graph G, $\text{PERM}(G)$ is the number of cycle-covers of G.

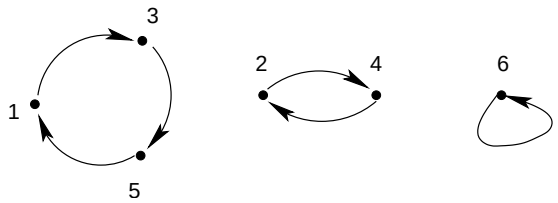

Figure 6.1: Directed graph corresponding to $(1\ 3\ 5)(2\ 4)(6)$

Proof Idea: We will reduce #3-SAT to 0-1PERM in two steps. Given any 3-SAT formula φ, in the first step, we will create a weighted directed graph G' (with small weights) such that

$$\text{PERM}(G') = 4^{3m} \cdot \#(\varphi)$$

where m is the number of clauses in φ. In second step, we will convert G' to an unweighted graph G such that $\text{PERM}(G') = (\text{PERM}(G) \bmod M)$, where M will only have polynomially many bits.

First, we will construct G' from φ. The construction will be via gadgets. The VARIABLE gadget is shown in Figure 6.2. All the edges have unit weights. Notice that it contains one dotted edge for every occurrence of the variable in φ. Each dotted edge will be replaced by a subgraph which will be described later. Any cycle-cover either contains all dotted edges corresponding to a positive occurrence (and all self-loops corresponding to negative occurrence) or vice versa.

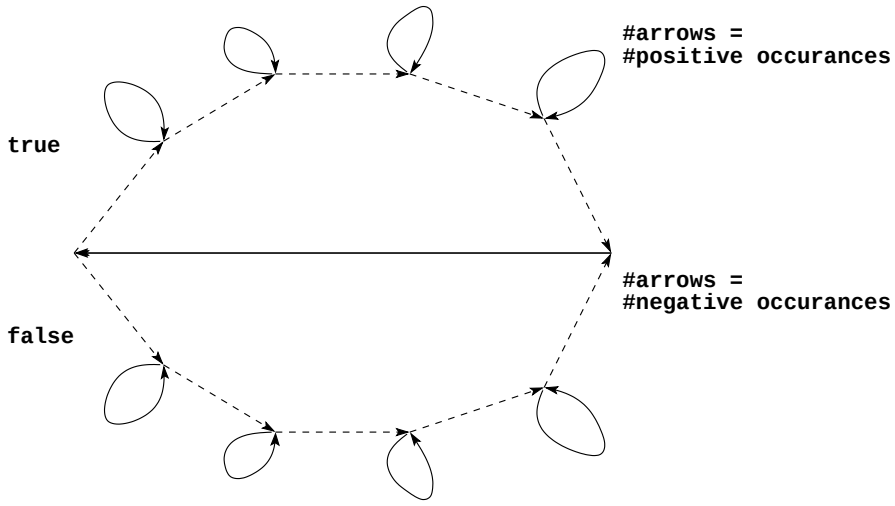

Figure 6.2: The VARIABLE gadget

The CLAUSE gadget is shown in Figure 6.3. It contains three dotted edges corresponding to three variables that occur in that clause. All the edges have unit weights. This gadget has the property that

A. in any cycle-cover, at least one of the dotted edges is not used, and

B. for any non-empty subset S of the dotted edges there is precisely one cycle-cover of the gadget that includes all dotted edges but those in S. (See Figure 6.4.)

Now, given any clause C and any literal x contained in it, there is a dotted edge (u, u') in the CLAUSE gadget for the literal and a dotted edge (v, v') in the appropriate side of VARIABLE gadget for the clause. These two dotted edges are replaced by an XOR gadget shown in Figure 6.5.

The XOR gadget has the property that the total contribution of all cycle-covers using none or both of (u, u') and (v, v') is 0. For cycle-covers using exactly one of the two, the gadget contributes a factor of 4. To see this, lets consider all possibilities:

A. None of the external edges is present: The cycle-covers are $(a\ c\ b\ d)$, $(a\ b)(c\ d)$, $(a\ d\ b)(c)$ and $(a\ d\ c\ b)$. The net contribution is (-2) + 6+ (-1) + (-3) = 0.

B. Precisely (u, a) and (a, v') are present: The cycle-covers are $(b\ c\ d)$, $(b\ d\ c)$, $(c\ d)(b)$ and $(c)(b\ d)$. The net contribution is (2) + (3)+ (-6) + (1) = 0.

C. Precisely (v, d) and (d, u') are present: The cycle-covers are $(a\ b)(c)$ and $(a\ c\ b)$. The net contribution is 1 + (-1) = 0.

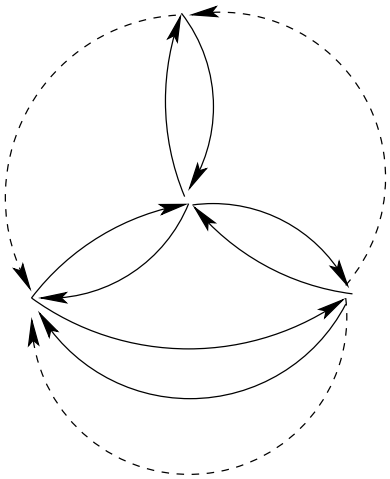

Figure 6.3: The CLAUSE gadget

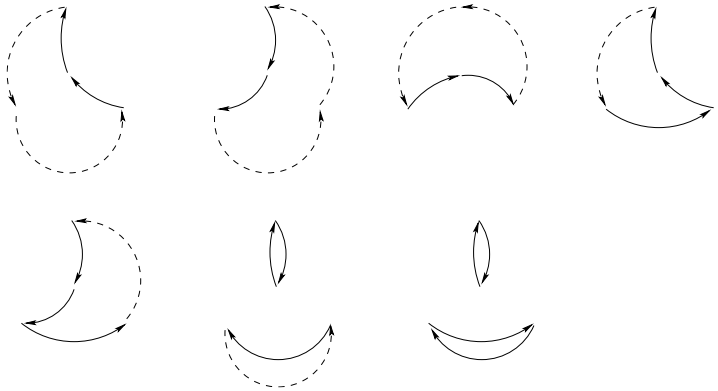

Figure 6.4: The cycle-covers of the CLAUSE gadget

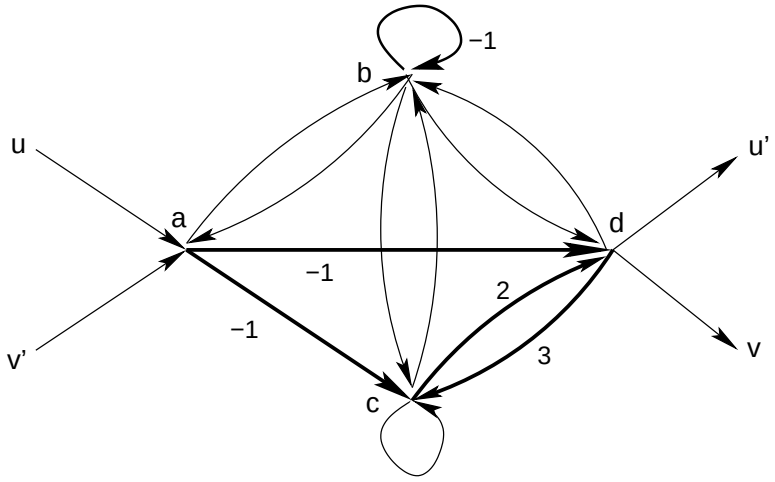

Figure 6.5: The XOR gadget

D. All four external edges are present: The cycle-covers are $(b\ c)$ and $(b)(c)$. The net contribution is 1 +

(-1) = 0.

E. Precisely (v, d) and (a, v') are present: In this case the gadget contains a path from d to a (represented with square brackets) as well as a cycle-cover involving the remaining vertices. The contributions to the cycle-covers are $[d\ b\ a](c)$ and $[d\ c\ b\ a]$. The net contribution is 1 + 3 = 4.

F. Precisely (u, a) and (d, v') are present: The cycle-covers are $[a\ d](b\ c)$, $[a\ d](b)(c)$, $[a\ b\ d](c)$, $[a\ c\ d](b)$, $[a\ b\ c\ d]$ and $[a\ c\ b\ d]$. The net contribution is (-1) + 1 + 1 + 2 + 2 + (-1) = 4.

There are $3m$ XOR gadgets. As a result, every satisfying assignment of truth values to φ will contribute 4^{3m} to the cycle-cover and every other assignment will contribute 0. Hence,

$$\text{PERM}(G') = 4^{3m}\#(\varphi)$$

Now, we will convert G' to an unweighted graph G. Observe that $\text{PERM}(G') \leq 4^{3m}2^n \leq 2^{6m+n}$. Let $N = 6m + n$ and $M = 2^N + 1$. Replace the weighted edges in G' with a set of unweighted edges as shown in Figure 6. For weights 2 and 3, the conversion does not affect the total weight of cycle-covers. For weight -1, the conversion blows up the total weight by $2^N \equiv -1(modM)$. As a result, if G is the resulting unweighted graph, $\text{PERM}(G') \equiv \text{PERM}(G) \pmod{M}$.

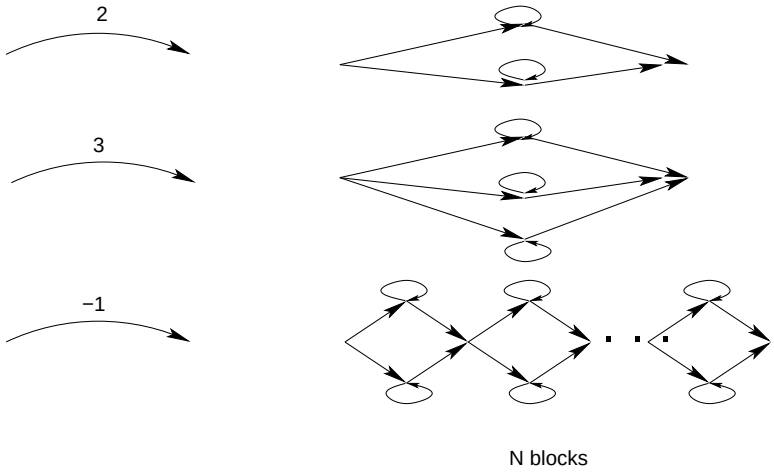

Figure 6.6: Conversion to an unweighted graph

Thus, we have shown a reduction of #3-SAT to 0-1PERM. This proves the theorem.

□

Definition 6.1. For a complexity class $\mathcal{C}$, define

$$\oplus\mathcal{C} = \{A | \exists R \in \mathcal{C},\ \text{polynomial bound}\ \ p : \mathbb{N} \to \mathbb{N}, s.t.\ x \in A \Leftrightarrow \#^{p(|x|)}y.(x, y) \in R\ \text{ is odd }\},$$

$$\mathsf{BP}\mathcal{C} = \{A | \exists R \in \mathcal{C},\ \text{polynomial bound}\ \ p : \mathbb{N} \to \mathbb{N}, s.t.\ \frac{\#^{p(|x|)}y.\ (x, y) \in R}{2^{p(|x|)}} \begin{cases} \geq \frac{2}{3} & \text{if } x \in A \\ \leq \frac{1}{3} & \text{if } x \notin A \end{cases}\}$$

Theorem 6.2 (Toda). $\mathsf{PH} \subseteq \mathsf{BP} \oplus \mathsf{P} \subseteq \mathsf{P}^{\#\mathsf{P}}$.

Before we prove Toda's theorem, we will prove the following theorem as a warm-up since it introduces most of the ideas.

Theorem 6.3 (Valiant-Vazirani, Toda). $\mathsf{NP} \subseteq \mathsf{R} \oplus \mathsf{P} \subseteq \mathsf{RP}^{\oplus \mathsf{P}}$.

Proof. We will concentrate on SAT for intuition. The following argument could apply to any NP language equally well. Let φ be any boolean formula and let S be the set of satisfying assignments to φ. To decide if φ is satisfiable we could imagine feeding φ to an oracle for $\oplus\mathrm{SAT} \in \oplus\mathsf{P}$. If $|S|$ is odd, then the $\oplus$SAT oracle will return *yes* and we know that φ is satisfiable and we are done. However if $|S| > 0$ is even then we would get back a *no* answer which would not be helpful.

The solution is to add random linear constraints in order to reduce $|S| > 0$ if necessary so that it is odd. In particular we will add linear constraints so that the remaining set of assignments S' has $|S'| = 1$ which is certainly odd.

Now we describe the notation for the random linear constraints. View $S \subseteq \{0,1\}^n = \mathbb{F}_2^n$ where $\mathbb{F}_2$ is the field with 2 elements.

Definition 6.2. For $v_1, v_2, \ldots, v_i \in \mathbb{F}_2^n$, let

$$< v_1, v_2, \ldots, v_i >^{\perp} = \{x \in \mathbb{F}_2^n | v_1 \cdot x = v_2 \cdot x = \ldots = v_i \cdot x = 0\}$$

Definition 6.3. For convenience of notation we write $a \in_R A$ to say that a is chosen uniformly at random from A and $a_1, \ldots, a_k \in_R A$ so say that $a_1, \ldots, a_k$ are chosen uniformly and independently at random from A.

Lemma 6.4 (Valiant-Vazirani). *If S is any non-empty subset of $\mathbb{F}_2^n$, then for $v_1, v_2 \cdots v_n \in_R \mathbb{F}_2^n$,*

$$Pr[\exists i \in \{1, \cdots n\}. \, |S \cap < v_1, v_2, \ldots, v_i >^{\perp} | = 1] \geq \frac{1}{4}.$$

We use the following weaker form which suffices:

Lemma 6.5. *If S is any non-empty subset of $\mathbb{F}_2^n$, then for $v_1, v_2 \cdots v_{n+1} \in_R \mathbb{F}_2^n$,*

$$Pr[\exists i \in \{1, \cdots n\}. \, |S \cap < v_1, v_2, \ldots, v_{i+1} >^{\perp} | = 1] > \frac{1}{8}.$$

We now see how this suffices to prove that $\mathsf{NP} \in \mathsf{RP}^{\oplus \mathsf{P}}$. Define φ_v such that $\varphi_v(x)$ is true if and only if $v \cdot x = 0$. Here is the algorithm:
Choose $v_1, \cdots v_n \in_R \mathbb{F}_2^n$
For each i from 1 to n, call a $\oplus$SAT oracle on $\varphi_i = \varphi \wedge \varphi_{v_1} \wedge \cdots \wedge \varphi_{v_{i+1}}$.
Accept iff one of the calls accepts.

Next class we will prove Lemma 6.5 and show the stronger result that $\mathsf{NP} \subseteq \mathsf{R}. \oplus \mathsf{P}$. □

Lecture 7

Toda's Theorem

April 20, 2004
Lecturer: Paul Beame
Notes: Tian Sang

Definition 7.1. If P is any predicate define

$$\begin{aligned} \oplus^k y.\ P(y) &= |\{y \in \{0,1\}^k : P(y)\}| \bmod 2, \\ \#^k y.P(y) &= |\{y \in \{0,1\}^k : P(y)\}| = \sum_{y \in \{0,1\}^k} P(y), \\ \text{Я}^k y.\ P(y) &= \frac{|\{y \in \{0,1\}^k : P(y)\}|}{2^k} \end{aligned}$$

Definition 7.2. For complexity class C, define

$$\begin{aligned} L \in \oplus \cdot \mathsf{C} \quad &\Leftrightarrow \quad \text{there is a relation } R \in \mathsf{C} \text{ and polynomial } p \text{ such that} \\ &\qquad L = \{x \mid \oplus^{p(|x|)}\ y.\ R(x,y)\} \\ L \in \mathsf{BP} \cdot \mathsf{C} \quad &\Leftrightarrow \quad \text{there is a relation } R \in \mathsf{C} \text{ and polynomial } p \text{ such that for some } \epsilon < 1/2 \\ &\qquad \begin{cases} \text{Я}^{p(|x|)} y.\ R(x,y) \geq 1-\epsilon & \text{for } x \in L \\ \text{Я}^{p(|x|)} y.\ R(x,y) \leq \epsilon & \text{for } x \notin L \end{cases} \\ L \in \mathsf{R} \cdot \mathsf{C} \quad &\Leftrightarrow \quad \text{there is a relation } R \in \mathsf{C} \text{ and polynomial } p \text{ such that for some } \epsilon < 1 \\ &\qquad \begin{cases} \text{Я}^{p(|x|)} y.\ R(x,y) \geq 1-\epsilon & \text{for } x \in L \\ \text{Я}^{p(|x|)} y.\ R(x,y) = 0 & \text{for } x \notin L \end{cases} \\ L \in \mathsf{P} \cdot \mathsf{C} \quad &\Leftrightarrow \quad \text{there is a relation } R \in \mathsf{C} \text{ and polynomial } p \text{ such that} \\ &\qquad L = \{x \mid \text{Я}^{p(|x|)} y.\ R(x,y) > 1/2\} \end{aligned}$$

Theorem 7.1 (Toda). $\mathsf{PH} \subseteq \mathsf{P}^{\#\mathsf{P}} = \mathsf{P}^{\mathrm{PERM}}$.

As outlined in the last lecture this is done in two steps.

Lemma 7.2 (Toda). $\mathsf{PH} \subseteq \mathsf{BP} \cdot \oplus\mathsf{P}$.

Lemma 7.3 (Toda). $\mathsf{BP} \cdot \oplus\mathsf{P} \subseteq \mathsf{P} \cdot \oplus\mathsf{P} \subseteq \mathsf{P}^{\#\mathsf{P}}$.

As a warm-up we finish the proof of the following lemma which provides the key ideas for the proof of Lemma 7.2.

Theorem 7.4 (Valiant-Vazirani, Toda). $\mathsf{NP} \subseteq \mathsf{R}.\oplus\mathsf{P} \subseteq \mathsf{RP}^{\oplus\mathsf{P}}$.

Proof. To prove Theorem 7.4, as outlined in the last lecture we convert an NP problem to equivalent problem for which there is precisely one solution (and therefore the number of solutions will be odd). This will be accomplished by adding randomly chosen linear constraints. We use the following slightly weakened version of a lemma due to Valiant and Vazirani.

Lemma 7.5 (Valiant-Vazirani). *If $\emptyset \neq S \subseteq \mathbb{F}_2^n$, then for $v_1, \cdots v_{n+1} \in_R \mathbb{F}_2^n$,*

$$\Pr[\exists i \in \{1, \cdots n\}.\ |S \cap \langle v_1, v_2, \ldots, v_{i+1}\rangle^\perp| = 1] > \frac{1}{8}.$$

This follows immediately from the following lemma.

Lemma 7.6. *Fix a set $S \subseteq \mathbb{F}_2^n$, then for $v_1, \ldots, v_{n+1} \in_R \mathbb{F}_2^n$,*

(a) if $0^n \in S$, then $\Pr[\ |S \cap \langle v_1, v_2, \ldots, v_{n+1}\rangle^\perp| = 1\ ] > \frac{1}{2}$

(b) if $0^n \notin S$, and $2^{i-1} \leq |S| \leq 2^i$ then $\Pr[\ |S \cap \langle v_1, v_2, \ldots, v_{i+1}\rangle^\perp| = 1\ ] > \frac{1}{8}$

Proof. We first show part (a). Since we always have $0^n \in \langle v_1, v_2, \ldots, v_{i+1}\rangle^\perp$, if $0^n \in S$ then $0^n \in S \cap < v_1, v_2, \ldots, v_i >^\perp$. For any $x \in \mathbb{F}_2^n$, if $x \neq 0^n$ we have for any j that $\Pr[\ v_j \cdot x = 0\ ] = 1/2$. Therefore, since the v_j are chosen independently, $\Pr[\ v_1 \cdot x = v_2 \cdot x = \ldots = v_{n+1} \cdot x = 0\ ] = \frac{1}{2^{n+1}}$. Thus

$$\begin{aligned}\Pr[\exists x \in S - \{0^n\}, x \in \langle v_1, v_2, \ldots, v_{n+1}\rangle^\perp] &\leq \sum_{x \in S - \{0^n\}} \Pr[x \in \langle v_1, v_2, \ldots, v_{n+1}\rangle^\perp] \\ &= \frac{|S| - 1}{2^{n+1}} < 1/2.\end{aligned}$$

It follows that with probability greater than $1/2$, 0^n is the only element of $S \cap \langle v_1, v_2, \ldots, v_{n+1}\langle^\perp$ which means that $\Pr[\ |S \cap \langle v_1, v_2, \ldots, v_{n+1}\rangle^\perp| = 1\ ] > 1/2$.

We now prove part (b). Suppose that $0^n \notin S$ and $2^{-1} \leq |S| \leq 2^i$. Define $h(x) = (v_1 \cdot x, \cdots, v_{i+1} \cdot x) \in \mathbb{F}_2^{i+1}$. As in the argument for part (a), for $x \neq 0^n$, $\Pr[h(x) = 0^{i+1}] = 1/2^{i+1}$. An alternative way of viewing this probability statement is to view the condition that $h(x) = 0$ as a system of linear equations whose variables are the coordinates of the v_j vectors and whose coefficients are given by the coordinates of x. For $x \neq 0$, each equation $v_j \cdot x = x \cdot v_j = 0$ adds an additional independent constraint and therefore the dimension of the solution space drops by 1 for each j. In total, there are $i+1$ linearly independent equations in the v_j so the solution space is a $2^{-(i+1)}$ fraction of all possible vectors.

Suppose now that $x \neq y$ and $x, y \neq 0^n$, Then the condition that $h(x) = h(y) = 0^{i+1}$ is given by $2(i+1)$ equations whose coefficients are given by the coordinates of x and y. Since $x \notin \{0^n, y\}$ and $y \neq 0^n$, x and y are linearly independent and thus all of the $2(i+1)$ equations given by $h(x) = h(y) = 0^{i+1}$ are linearly independent. Therefore $\Pr[h(x) = h(y) = 0^{i+1}] = 1/2^{2(i+1)}$ for $x, y \in S$, $x \neq y$. Thus

$$\begin{aligned}\Pr[\exists y \in S - \{x\}.\ h(x) = h(y) = 0^{i+1}] &\leq \sum_{y \in S - \{x\}} \Pr[h(x) = h(y) = 0^{i+1}] \\ &= \frac{|S| - 1}{2^{2(i+1)}} \\ &< \frac{1}{2^{i+2}} \qquad \text{since } |S| \leq 2^i.\end{aligned}$$

Therefore

$$\begin{aligned}
&\Pr[h(x) = 0^{i+1} \text{ and } \forall y \in S - \{x\}.\ h(y) \neq 0^{i+1}] \\
&\quad = \Pr[h(x) = 0^{i+1}] - \Pr[\exists y \in S - \{x\}.\ h(x) = h(y) = 0^{i+1}] \\
&\quad > \frac{1}{2^{i+1}} - \frac{1}{2^{i+2}} = \frac{1}{2^{i+2}}.
\end{aligned}$$

Taking the union of these events, which are disjoint, over all choices of $x \in S$,

$$\begin{aligned}
\Pr[\exists x.\ h(x) = 0^{i+1} \text{ and } \forall y \in S - \{x\}.\ h(y) \neq 0^{i+1}] &> \frac{|S|}{2^{i+2}} \\
&\geq \frac{2^{i-1}}{2^{i+2}} = \frac{1}{8} \qquad \text{since } |S| \geq 2^{i-1}
\end{aligned}$$

as required. □

We now prove Theorem 7.4 that $\mathsf{NP} \subseteq \mathsf{R} \cdot \oplus\mathsf{P}$. The key difference between showing this and showing that $\mathsf{NP} \in \mathsf{RP}^{\oplus\mathsf{P}}$ is the requirement that the algorithm make only one query to the $\oplus\mathsf{P}$ oracle and return the answer as its output. In order to do this we begin with a different basic experiment from the one outlined at the end of last lecture. Instead of trying all possible values of i we choose i at random. With probability at least $1/n$ this choice of i will be the correct i for Lemma 7.5. Here is the basic experiment E on input φ:

A. choose $i \in_R \{1, \ldots, n\}$, and $v_1, \ldots, v_{i+1} \in_R \mathbb{F}_2^n$

B. query the $\oplus$SAT oracle on $\varphi \wedge \varphi_{v_1} \wedge \cdots \wedge \varphi_{v_{i+1}}$ where φ_v is a formula that is satisfied by x iff $v \cdot x = 0$.

C. accept iff the oracle accepts.

Observe that if φ has n variables then

- if φ is unsatisfiable then $\Pr[\ E \text{ accepts } \varphi] = 0$, and
- if φ is satisfiable then $\Pr[\ E \text{ accepts } \varphi] > \frac{1}{8n}$.

This yields a randomized algorithm (with one call to an $\oplus\mathsf{P}$ oracle) for SAT with 1-sided error but its success probability is too low. To boost its success probability we make m independent trials of E. Each trial chooses an integer i in $\{1, \ldots, m\}$ and sequence of $i + 1$ vectors.

Let $r_1 = (i_1, v_1^1, \ldots, v_{i_1}^1)$ through $r_m = (i_1, v_1^m, \ldots, v_{i_m}^m)$ be the sequence of random choices of the independent trials of experiment E. Therefore

$$\Pr[\text{ all } m \text{ experiments fail}] \leq (1 - \frac{1}{8n})^m \leq e^{-\frac{m}{8n}} \leq e^{-2n} \qquad \text{for } m = 16n^2.$$

Thus

$$\varphi \in \text{SAT} \Leftrightarrow \text{Я} r_1, \ldots, r_m \underbrace{\exists j \in [16n^2].\ \varphi_{r_j} \in \oplus SAT}\ \geq 1 - e^{-2n}$$

where φ_{r_j} is the variant of φ created by added the linear constraints given by random string r_j. In comparison with trying all values of i, this does not seem to have helped much (except to reduce the error) because there are now $16n^2$ calls to the $\oplus\mathsf{P}$ oracle. We will concentrate on modifying the part of formula marked with the underbrace so that only one such call is required.

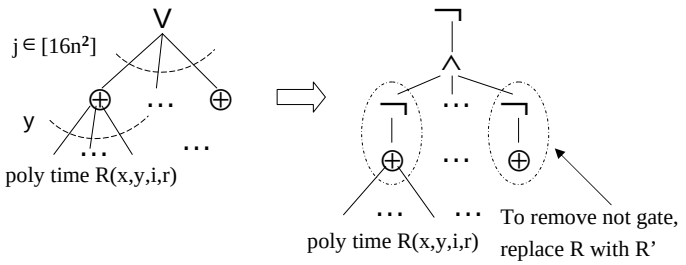

Figure 7.1: Converting from OR (∃) to AND (∀

We now express things more generally using the definition of $\oplus\mathsf{P}$. We have shown that for any NP language L, there is a polynomial-time computable relation R and a polynomial bound p such that

$$x \in L \Leftrightarrow \text{Я}r \underbrace{\exists j \in [16N^2] \oplus^N y. R(x,y,j,r)} \geq 1 - e^{-2N}$$

where $N = p(|x|)$ and $r = (r_1, \ldots, r_{16N^2})$. Now

$$\begin{aligned} \exists j \in [16N^2] \oplus^N y.\ R(x,y,j,r) &= \neg\forall j \in [16N^2] \neg \oplus^N y.\ R(x,y,j,r) \\ &= \neg\forall j \in [16N^2] \oplus^{N+1} y.\ \overline{R}(x,y,j,r) \end{aligned}$$

where $\overline{(R)}$ is defined as in the following lemma.

Lemma 7.7. *If $A \in \oplus\mathsf{P}$ then $\overline{A} \in \oplus\mathsf{P}$.*

Proof. We simply add an additional 1 input to the parity quantifier to negate its value. More precisely, if $x \in A \Leftrightarrow \oplus^N y.\ R(x,y)$ for polynomial-time computable R then $x \in \overline{A} \Leftrightarrow \oplus^{N+1} y'.\ \overline{R}(x,y')$ where $\overline{R}(x,y') = ((y' = 0^{N+1}) \vee ((y' = 1y) \wedge R(x,y)))$. □

Since an $\exists$ quantifier acts as an OR and a $\forall$ acts as an AND we can view this as in Figure 7.1.

Since its inputs are 0-1-valued, the $\forall$ (or AND) acts simply as a fan-in $16N^2$ multiplication of large fan-in sums modulo 2. Expanding out this product of sums as a sum of products yields

$$\begin{aligned} \neg\forall j \in [16N^2] \oplus^{N+1} y.\ \overline{R}(x,y,j,r) &= \neg \oplus^{16N^2(N+1)} y_1, \ldots, y_{16N^2} \bigwedge_{j=1}^{16N^2} \overline{R}(x,y_j,j,r) \\ &= \neg \oplus^{16N^2(N+1)} y R'(x,y,j,r) \qquad \text{for some polytime } R' \\ &= \oplus^{16N^2(N+1)+1} y \overline{R'}(x,y,j,r) \qquad \text{incorporating the negation.} \end{aligned}$$

This is only a single call to a $\oplus\mathsf{P}$ oracle. Plugging this in for the quantity in the underbrace yields the desired $\mathsf{R} \cdot \oplus\mathsf{P}$ algorithm. □

This argument has yielded almost all the ideas we will need for proving Lemma 7.2.

Proof of Lemma 7.2. Suppose $L \in \mathsf{PH}$, then there is some k such that $L \in \Sigma_k\mathsf{P}$. Therefore there is some relation R and polynomial p such that

$$\begin{aligned} L &= \{x \mid \exists^{p(|x|)} y_1 \forall^{p(|x|)} y_2 \exists \ldots Q^{p(|x|)} y_k.\ R(x,y_1,\ldots,y_k)\} \\ &= \{x \mid \exists^{p(|x|)} y_1 \neg\exists^{p(|x|)} y_2 \neg\exists\neg \ldots Q^{p(|x|)} y_k.\ R(x,y_1,\ldots,y_k)\}. \end{aligned}$$

Consider expanding the tree of quantifiers as a giant tree of possible values. For each of the $2^{jp(|x|)}$ values of the prefix $y_1, \ldots, y_j$ for $0 \leq j \leq k-1$ there is an $\exists$ node in the tree. The total number of such nodes over all values of $j \leq k-1$ is less than $2^{kp(|x|)}$. Let $N = kp(|x|)$. Choose $16N^2$ tuples $r_j = (i_j, v_1^j, \ldots, v_{i_j+1}^j)$ where $i_j \in [N]$ and $v_i^j \in \mathbb{F}_2^N$ as in the proof of Theorem 7.4. Apply the argument of Theorem 7.4 (before the conversion to a single $\oplus\mathsf{P}$ call) simultaneously to all the predicates corresponding to the all $\exists$ nodes, using the same sequence of random choices. (This involves adding the same set of linear constraints to augment each of the $\exists$ nodes in this tree.) For a fixed input x and a fixed node, the probability that the value at that node is incorrectly computeed is at most e^{-2N}. There are fewer than 2^N nodes in the tree and only 2^n inputs x of length n. Therefore the probability that there is some node of the tree that is computed incorrectly is at most $2^n \cdot 2^N \cdot e^{-2N} < \frac{1}{4}$.

So we have an computation for $x \in L$ described as

$$\text{Я}\overrightarrow{r}\exists j_1 \in [16N^2] \oplus^N y_1 \cdots \exists j_k \in [16N^2] \oplus^N y_k.\ R(x, \overrightarrow{r}, j_1, \ldots, j_k, y_1, \ldots, y_k)$$

for some polynomial-time computable relation R that gives the correct value all but at most $1/4$ of the time (over the random choices r). This is a bounded-error algorithm for L. (Note that because of the negations in the tree, the error is 2-sided and no longer 1-sided as in the case of NP.)

Again we multiply out the small fan-in $\exists$ quantifiers to yield to rewrite this expression as:

$$\text{Я}\overrightarrow{r} \oplus^{(16N^2)^k N} (y_1^1 \ldots y_k^1) \ldots (y_1^{16N^2}, \ldots, y_k^{16N^2}) \bigwedge_{j_1 \ldots j_k \in [16N^2]^k} \overline{R}(x, \overrightarrow{r}, j_1, \ldots, j_k, y_1^{j_1}, \ldots, y_k^{j_k}).$$

Since k is constant, this vector of y values has polynomial size and the interior computation is polynomial time, and thus $L \in \mathsf{BP} \cdot \oplus\mathsf{P}$. □

Proof of Lemma 7.3. We show that $\mathsf{BP} \cdot \oplus\mathsf{P} \subseteq \mathsf{P}^{\#\mathsf{P}}$. Suppose that $L \in \mathsf{BP} \cdot \oplus\mathsf{P}$. (It will be easy to see that the result also holds for the unbounded-error complexity class $\mathsf{P} \cdot \oplus\mathsf{P}$.) Then there is some polynomial-time TM M and polynomial function $N = p(|x|)$, such that

$$\begin{aligned} L &= \{x \mid \text{Я}^N r \oplus^N y.\ M(x, r, y) > 1/2\} \\ &= \{x \mid \sum_{r \in \{0,1\}^N} B(x, r) > 2^{N-1}\}, \end{aligned}$$

where $B(x, r) = \sum_{y \in \{0,1\}^N} M(x, r, y) \bmod 2$

Here is the basic proof idea: Suppose we could create a polynomial time TM M' such that

$$\sum_{y' \in \{0,1\}^{N'}} M'(x, y', r) \equiv B(x, r) \pmod{2^{N+1}}.$$

Then we would have $\sum_{r \in \{0,1\}^N} B(x, r) \equiv \sum_{r \in \{0,1\}^N} \sum_{y' \in \{0,1\}^{N'}} M'(x, y'r) \pmod{2^{N+1}}$.

Then to tell whether or not $x \in L$, we can simply compute $\#^{N+N'}(r, y').\ M'(x, y', r)$ using the $\#\mathsf{P}$ oracle, take that value $\bmod 2^{N+1}$, and accept if the result is $> 2^{N-1}$.

We will not quite be able to do this but we will be able to find an M' such that

$$\sum_{y' \in \{0,1\}^{N'}} M'(x, y', r) \equiv -B(x, r) \pmod{2^{N+1}}.$$

By a similar argument we can decide L by making a call to compute $\#^{N+N'}(r, y')$. $M'(x, y', r)$ using the $\#\mathsf{P}$ oracle, taking that value $\bmod 2^{N+1}$, and accepting if the result is $< 2^{N+1} - 2^{N-1}$.

In order to satisfy these conditions we need to convert the number of accepting computations from being either 0 or -1 modulo 2 into a number that is either 0 or -1 modulo 2^{N+1}. The key technical lemma is the following:

Lemma 7.8. *For integers a and $m > 0$,*

(a) if $a \equiv 0 \pmod m$ then $4a^3 + 3a^4 \equiv 0 \pmod{m^2}$, and

(b) if $a \equiv -1 \pmod m$ then $4a^3 + 3a^4 \equiv -1 \pmod{m^2}$.

Proof. Part (a) follows because $m^2|a^2$ and $a^2|4a^3 + 3a^4$.

For part (b) write $a = km - 1$. Then

$$\begin{aligned} 4a^3 + 3a^4 &= 4(km-1)^3 + 3(km-1)^4 \\ &\equiv 12km - 4 + (-12km + 3) \pmod{m^2} \\ &\equiv -1 \pmod{m^2}. \end{aligned}$$

□

We apply this lemma inductively to construct polynomial-time machines $M_i(x, y, r)$ such that $\sum_y M_i(x, y, r) \equiv -B(x, r) \pmod{2^{2^i}}$. Applying the construction until $i = \lceil \log_2(N + 1) \rceil$ will yield the desired M'. For the base case $i = 0$, choosing $M_0(x, y, r) = M(x, y, r)$ we have $\sum_y M_0(x, y, r) \equiv -B(x, r) \pmod 2$ as required. For the inductive step, suppose that we have already constructed M_i. We will apply Lemma 7.8 with $m = 2^{2^i}$ and note that $m^2 = 2^{2^{i+1}}$. We create a new machine M_{i+1} so that on input (x, r) if a is the number of accepting choices for y in M_i then M_{i+1} will have $4a^3 + 3a^4$ accepting choices of its corresponding y'.

Let $y' = (z_1, z_2, z_3, y_1, y_2, y_3, y_4)$, such that $z_i \in \{0, 1\}$, and $y_i \in \{0, 1\}^{|y|}$. Define

$$\begin{aligned} M_{i+1}(x, r, y') &= (z_1 \wedge M_i(x, r, y_1) \wedge M_i(x, r, y_2) \wedge M_i(x, r, y_3) \wedge (y_4 = 0^{|y|})) \\ &\quad \vee ((\overline{z_1} \wedge (z_2 \vee z_3)) \wedge M_i(x, r, y_1) \wedge M_i(x, r, y_2) \wedge M_i(x, r, y_3) \wedge M_i(x, r, y_4)). \end{aligned}$$

It is easy to see the M_{i+1} has the desired number of accepting choices of y' as a function of the number of choices of y for M_i. By Lemma 7.8, we know that $\sum_{y'} M_{i+1}(x, y', r) \equiv -B(x, r) \pmod{2^{2^{i+1}}}$.

It remains to confirm the final machine M' is polynomial-time computable. It clearly will be polynomial in the length of the final string y'. Since the number of times to repeat this construction is $i = \lceil \log_2(N+1) \rceil$, and at each iteration $|y'| = 3 + 4|y|$, $|y'|$ grows by factor ≤ 7 per iteration. Therefore the total length of y' at the end is $\leq 7^{\log_2(N+1)} N$, which is polynomial in N and therefore polynomial in $|x|$ as required.

We now simply fit this into the framework to yield a polynomial-time algorithm for L that makes a single call to a $\#\mathsf{P}$ oracle. □

Lecture 8

Time-Space Tradeoffs for SAT

April 22, 2004
Lecturer: Paul Beame
Notes: Geoff Mohr

Definition 8.1. $\mathsf{TIMESPACE}(T(n), S(n)) = \{L \in \{0,1\}^* \mid \exists$ offline, multitape TM M such that $L = L(M)$ and M uses time $O(T(n))$ and space $O(S(n))\ \}$.

One should note that, while $\mathsf{TIMESPACE}(T(n), S(n)) \subseteq \mathsf{TIME}(T(n)) \cap \mathsf{SPACE}(S(n))$, in general the definition is different from $\mathsf{TIME}(T(n)) \cap \mathsf{SPACE}(S(n))$, since a *single* machine is required to run in both time $O(T(n))$ and space $O(S(n))$.

We will prove a variant of the following theorem which is the strongest form of time-space tradeoffs currently known. The first result of this sort was shown by Fortnow; on the way to showing bounds like these we prove bounds for smaller running times due to Lipton and Viglas.

Theorem 8.1 (Fortnow-van Melkebeek). *Let $\phi = (\sqrt{5}+1)/2$. There is a function $s : \mathbb{R}^+ \to \mathbb{R}^+$ s.t. for all $c < \phi$, $\mathsf{SAT} \notin \mathsf{TIMESPACE}(n^c, n^{s(c)})$ and $\lim_{c\to 1} s(c) = 1$.*

We will prove a weaker form of this theorem that for $c < \phi$, $\mathsf{SAT} \notin \mathsf{TIMESPACE}(n^c, n^{o(1)})$. By the efficient reduction described in Lemma 5.4 of $\mathsf{NTIME}(T(n))$ to 3-SAT, which maps an $\mathsf{NTIME}(T(n))$ machine to a 3CNF formula of size $O(T(n)\log T(n))$ in time $O(T(n)\log T(n))$ time and $O(\log T(n))$ space, to prove this weaker form it suffices to show the following.

Theorem 8.2. *For $c < (\sqrt{5}+1)/2$, $\mathsf{NTIME}(n) \not\subseteq \mathsf{TIMESPACE}(n^c, n^{o(1)})$.*

One simple and old idea we will use is that of *padding* which shows that if simulations at low complexity levels exist then simulations at high complexity levels also exist.

Lemma 8.3 (Padding Meta-lemma). *Suppose that $\mathsf{C}(n) \subseteq \mathsf{C}'(f(n))$ for parameterized complexity class families C and C' and some function $f : \mathbb{N} \to \mathbb{R}^+$. C, C' parameterized by n and $f(n)$ respectively. Then for any $t(n) \geq n$ such that is computable using the resources allowed for $\mathsf{C}(t(n))$, $\mathsf{C}(t(n)) \subseteq \mathsf{C}'(f(t(n)))$.*

Proof. Let $L \in \mathsf{C}(t(n))$ and let M be a machine deciding L that witnesses this fact. Let $x \in \{0,1\}^n$ be input for M. Pad x to length $t(n)$, say to creat $x^{pad} = x01^{t(n)-n-1}$. For $A \in \mathsf{C}(t(n))$ let $A^{pad} = \{x01^{t(|x|)-|x|-1} \mid x \in A\ \}$. Since it is very easy to strip off the padding, $A^{pad} \in \mathsf{C}(n)$; so, by assumption, there is a witnessing machine M' showing that $A^{pad} \in \mathsf{C}'(f(n))$. We use M' as follows: On input x, create x^{pad} and run M' on x^{pad}. The resources used are those of $\mathsf{C}'(f(n))$ as required. □

Theorem 8.4 (Seiferas-Fischer-Meyer). *If $f, g : \mathbb{N} \to \mathbb{N}$, f, g are time constructible, and $f(n+1)$ is $o(g(n))$ then $\mathsf{NTIME}(f(n)) \subsetneq \mathsf{NTIME}(g(n))$.*

Note that, unlike the case for the deterministic time hierarchy, there is no multiplicative $O(\log T(n))$ factor in the separation. In this sense, it is stronger at low complexity levels. However for very quickly growing functions (for example double exponential), the condition involving the $+1$ is weaker than a multiplicative $\log T(n)$. When we use it we will need to ensure that the time bounds we consider are fairly small.

Proof Sketch. The general idea of the simulation begins in a similar way to the ordinary time hierarchy theorem: One adds a clock and uses a fixed simulation to create a universal Turing machine for all machines running in time bound $f(n)$. In this case, unlike th deterministic case we can simulate an arbitrary k-tape NTM by a 2-tape NTM with no more than a constant factor simulation loss. This allows one to add the clock with no complexity loss at all. The hard part is to execute the complementation part of the diagonalization argument and this is done by a sophisticated padding argument that causes the additional $+1$. It uses the fact that unary languages of arbitrarily high NTIME complexity exist. □

We will use the following outline in the proof of Theorem 8.2.

A. Assume $\mathsf{NTIME}(n) \subseteq \mathsf{TIMESPACE}(n^c, n^{o(1)})$ for some constant c.

B. Show that for some (not too large) $t(n)$ that is time computable, $\mathsf{NTIME}(n) \subseteq \mathsf{TIME}(n^c) \implies \mathsf{TIMESPACE}(t(n), t(n)^{o(1)}) \subseteq \mathsf{NTIME}(t(n)^{f(c)+o(1)})$ for some constant $f(c) > 0$.

C. For $T(n) = t(n)^{1/c}$ where $t(n)$ is given in part B, put the two parts together derive

$$\begin{aligned} \mathsf{NTIME}(T(n)) &\subseteq \mathsf{TIMESPACE}(T(n)^c, T(n)^{o(1)}) && \text{from part A by padding Lemma 8.3} \\ &\subseteq \mathsf{NTIME}(T(n)^{cf(c)+o(1)}) && \text{by part B.} \end{aligned}$$

D. If $cf(c) < 1$, this is a contradiction to the NTIME hierarchy given in Theorem 8.4.

Part B of this outline requires the most technical work. The key notions in the proof involve extensions of the ideas behind Savitch's Theorem and for this it is convenient to use alternating time complexity classes.

Definition 8.2. Define $\Sigma_k\mathsf{TIME}(T(n))$ to be the set of all languages L such that $x \in L$ iff $\exists^{T(|x|)}y_1\forall^{T(|x|)}y_2 \ldots Q^{T(|x|)}y_k M(x, y_1, \ldots, y_k)$ where M runs for at most $O(T(|x|))$ steps on input $(x, y_1, \ldots, y_n)$. Define $\Pi_k\mathsf{TIME}(T(n))$ similarly.

Lemma 8.5. *For $S(n) \geq \log n$ and any integer function $b : \mathbb{N} \to \mathbb{N}$,*
$\mathsf{TIMESPACE}(T(n), S(n)) \subseteq \Sigma_2\mathsf{TIME}(T'(n))$ *where* $T'(n) = b(n) \cdot S(n) + T(n)/b(n) + \log b(n)$.

Proof. Recall from the proof of Savitch's Theorem that we can consider the computation as operating on the graph of TM configurations. For configurations C and C' we write $C \vdash^t C'$ if and only if configuration C yields C' in at most t steps. In the proof of Savitch's theorem we used the fact that we could assume a fixed form for the initial configuration C_0 and a unique accepting configuration C_f, and expressed

$$(C_0 \vdash^T C_f) \iff \exists C_m.\, ((C_0 \vdash^{T/2} C_m) \wedge (C_m \vdash^{T/2} C_f)).$$

In a similar way we can break up the computation into $b = b(n)$ pieces for any $b \geq 2$, so that, denoting C_f by C_b, we derive

$$(C_0 \vdash^T C_b) \iff \exists^{(b-1)S} C_1, C_2, \ldots, C_{b-1} \forall^{\log b} i.\, (C_{i-1} \vdash^{T/b} C_i).$$

Each configuration has size $O(S(n) + \log n) = O(S(n))$ and determining whether or not $C_{i-1} \vdash^{t/b} C_i$ requires time at most $O(T(n)/b(n) + S(n))$. Thus the computation overall is in $\Sigma_2\mathsf{TIME}(b(n) \cdot S(n) + T(n)/b(n) + \log b(n))$. □

Corollary 8.6. *For all* $T, S : \mathbb{N} \to \mathbb{R}^+$,
$\mathsf{TIMESPACE}(T(n), S(n)) \subseteq \Sigma_2\mathsf{TIME}(\sqrt{T(n)S(n)})$.

Proof. Apply Lemma 8.5 with $b(n) = \sqrt{\frac{T(n)}{S(n)}}$. □

Given a simple assumption about the simulation of nondeterministic time by deterministic time we see that we can remove an alternations from the computation.

Lemma 8.7. *If* $T(n) \geq n$ *is time constructible and* $\mathsf{NTIME}(n) \subseteq \mathsf{TIME}(T(n))$*, then for time constructible* $T'(n) \geq n, \Sigma_2\mathsf{TIME}(T'(n)) \subseteq \mathsf{NTIME}(T(T'(n)))$.

Proof. By definition, If $L \in \Sigma_2\mathsf{TIME}(T'(n))$ then there is some predicate R such that

$$x \in L \iff \exists^{T'(|x|)} y_1 \forall^{T'(|x|)} y_2 R(x, y_1, y_2)$$

and $R(x, y_1, y_2)$ is computable in time $O(T'(|x|))$. Therefore

$$x \in L \iff \exists^{T'(|x|)} y_1 \neg \exists^{T'(|x|)} y_2 \neg R(x, y_1, y_2).$$

By padding using the assumption that $\mathsf{NTIME}(n) \subseteq \mathsf{TIME}(T(n))$, we obtain $\mathsf{NTIME}(T'(n)) \subseteq \mathsf{TIME}(T(T'(n)))$ and thus the set

$$S = \{(x, y_1) \mid |y_1| \leq T'(|x|) \text{ and } \exists^{T'(|x|)} y_2 \neg R(x, y_1, y_2)\}$$

is in $\mathsf{TIME}(T(T'(n)))$. Since $x \in L$ if and only if $\exists^{T'(|x|)} y_1 \neg((x, y_1) \in S)$, it follows that

$$L \in \mathsf{NTIME}(T'(n) + T(T'(n))) = \mathsf{NTIME}(T(T'(n)))$$

as required. □

Note that the assumption in Lemma 8.7 can be weakened to $\mathsf{NTIME}(n) \subseteq \mathsf{coNTIME}(T(n))$ and the argument will still go through. We now obtain a simple version of Part B of our basic outline for the proof of Theorem 8.2.

Corollary 8.8. *If* $\mathsf{NTIME}(n) \subseteq \mathsf{TIME}(n^c)$ *then for* $t(n) \geq n^2$,
$\mathsf{TIMESPACE}(t(n), t(n)^{o(1)}) \subseteq \mathsf{NTIME}(t(n)^{c/2+o(1)})$.

Proof. By Corollary 8.6,

$$\mathsf{TIMESPACE}(t(n), t(n)^{o(1)}) \subseteq \Sigma_2\mathsf{TIME}(t(n)^{1/2+o(1)}).$$

Applying Lemma 8.7 with $T'(n) = t(n)^{1/2+o(1)} \geq n$ yields

$$\Sigma_2\mathsf{TIME}(t(n)^{1/2+o(1)}) \subseteq \mathsf{NTIME}(t(n)^{c/2+o(1)})$$

as required since $(t(n)^{o(1)})^c$ is $t(n)^{o(1)}$. □

Corollary 8.9 (Lipton-Viglas). $\mathsf{NTIME}(n) \not\subseteq \mathsf{TIMESPACE}(n^c, n^{o(1)})$ *for* $c < \sqrt{2}$.

Proof. Let $t(n)$ be as defined in Corollary 8.8 and let $T(n) = t(n)^{1/c}$. If $\mathsf{NTIME}(n) \subseteq \mathsf{TIMESPACE}(n^c, n^{o(1)})$ then

$$\begin{aligned}
\mathsf{NTIME}(T(n)) &\subseteq \mathsf{TIMESPACE}(T(n)^c, T(n)^{o(1)}) \\
&= \mathsf{TIMESPACE}(t(n), t(n)^{o(1)}) \\
&\subseteq \mathsf{NTIME}(t(n)^{c/2+o(1)}) \qquad \text{by Corollary 8.8} \\
&= \mathsf{NTIME}(T(n)^{c^2/2+o(1)}).
\end{aligned}$$

Since $c < \sqrt{2}$, $c^2/2 < 1$ and this yields a contradiction to the nondeterministic time hierarchy Theorem 8.4. □

Fortnow and van Melkebeek derived a slightly different form from Lemma 8.5 for alternating time simulation of $\mathsf{TIMESPACE}(T(n), S(n))$ using the following idea. For a deterministic Turing machine running for T steps we know that $C'_0 \vdash^T C_b$ if and only for all $C'_b \neq C_b$, $C'_0 \nvdash^T C'_b$. Furthermore Therefore

$$(C'_0 \vdash^T C_b) \iff \forall^{bS} C'_1, C'_2, \ldots, C'_{b-1}, C'_b \exists^{\log b} i.\ ((C'_b = C_b) \vee (C'_{i-1} \nvdash^{T/b} C'_i)).$$

Strictly speaking this construction would allow one to derive the following lemma.

Lemma 8.10. *For $S(n) \geq \log n$ and any integer function $b : \mathbb{N} \to \mathbb{N}$,*
$\mathsf{TIMESPACE}(T(n), S(n)) \subseteq \Pi_2\mathsf{TIME}(T'(n))$ *where* $T'(n) = b(n) \cdot S(n) + T(n)/b(n) + \log b(n)$.

This is no better than Lemma 8.5 but we will not use the idea in this simple form. The key is that we will be able to save because we have expressed $C'_0 \vdash^T C_b$ in terms of $C'_{i-1} \nvdash^{T/b} C'_i$. This will allow us to have fewer alternations when the construction is applied recursively since the $\neg\forall C'_1, \ldots$ quantifiers that will occur at the next level can be combined with the $\exists i$ quantifier at the current level. This is the idea behind the proof of Theorem 8.2.

Proof of Theorem 8.2. We first prove the following by induction on k. Let f_k be defined by $f_{k+1}(c) = c \cdot f_k(c)/(1 + f_k(c))$ and $f_1(c) = c/2$.

Claim 1. If $\mathsf{NTIME}(n) \subseteq \mathsf{TIME}(n^c)$ then for $k \geq 1$ and some not too large $t(n)$,

$$\mathsf{TIMESPACE}(t(n), t(n)^{o(1)}) \subseteq \mathsf{NTIME}(t(n)^{f_k(c)+o(1)}).$$

Proof of Claim. The base case $k = 1$ is precisely Corollary 8.8. Suppose that the claim is true for k. Suppose that we have a machine M witness a language L in $\mathsf{TIMESPACE}(t(n), t(n)^{o(1)})$. We apply the following expansion.

$$(C'_0 \vdash^t C_b) \iff \forall^{bS} C'_1, C'_2, \ldots, C'_{b-1}, C'_b \exists^{\log b} i.\ ((C'_b = C_b) \vee (C'_{i-1} \nvdash^{t/b} C'_i)).$$

Choose $b(n) = t(n)^{f_k(c)/(1+f_k(c)}$. Then $t(n)/b(n) = t(n)^{1/(f_k(c)+1)}$ and $b(n)s(n) = t(n)^{f_k(c)/(1+f_k(c))+o(1)}$. Since $f_k(c) \leq f_1(c) = c/2 < 1$, $b(n)s(n) \leq t(n)/b(n)$ so the computation time for the expansion is dominated by $t(n)/b(n) = t(n)^{1/(f_k(c)+1)}$. By the inductive hypothesis applied to the inner $C'_{i-1} \nvdash^{t/b} C'_i$ we obtain that for some not too large $t(n)/b(n)$ this computation can be done in

$$\mathsf{NTIME}([t(n)/b(n)]^{f_k(c)+o(1)}) = \mathsf{NTIME}(t(n)^{f_k(c)/(f_k(c)+1)+o(1)}).$$

Adding the $\exists^{\log b} i$ also keeps it in the same complexity class. By padding and the hypothesis that $\mathsf{NTIME}(n) \subseteq \mathsf{TIME}(n^c)$ we obtain that the inner computation $\exists^{\log b} i.\ ((C'_b = C_b) \vee (C'_{i-1} \not\vdash^{t/b} C'_i))$ can be done in

$$\mathsf{TIME}(t(n)^{c \cdot f_k(c)/(f_k(c)+1)+o(1)}).$$

Plugging this in we obtain that

$$\mathsf{TIMESPACE}(t(n), t(n)^{o(1)}) \subseteq \mathsf{coNTIME}(t(n)^{c \cdot f_k(c)/(f_k(c)+1)+o(1)}).$$

Since $\mathsf{TIMESPACE}(t(n), t(n)^{o(1)})$ is closed under complement and by the definition of $f_{k+1}(c)$ we obtain that

$$\mathsf{TIMESPACE}(t(n), t(n)^{o(1)}) \subseteq \mathsf{NTIME}(t(n)^{f_{k+1}(c)+o(1)})$$

as required. □

Applying the end of the proof outline we obtain that for any k, if $\mathsf{NTIME}(n) \subseteq \mathsf{TIMESPACE}(n^c, n^{o(1)})$ then for $T(n) = t(n)^{1/f_k(c)}$,

$$\begin{aligned}
\mathsf{NTIME}(T(n)) &\subseteq \mathsf{TIMESPACE}(T(n)^c, T(n)^{o(1)}) \\
&= \mathsf{TIMESPACE}(t(n), t(n)^{o(1)}) \\
&\subseteq \mathsf{NTIME}(t(n)^{f_k(c)+o(1)}) \\
&= \mathsf{NTIME}(T(n)^{c \cdot f_k(c)+o(1)}).
\end{aligned}$$

Observe that $f_k(c)$ is a monotonically decreasing function with fixed point $f_*(c) = c \cdot f_*(c)/(f_*(c)+1)$ when $f_*(c) = c-1$. Then $c \cdot f_*(c) = c(c-1) < 1$ when $c < \phi = (\sqrt{5}+1)/2$ which provides a contradiction to the nondeterministic time hierarchy theorem. □

Open Problem 8.1. Prove that $\mathsf{NTIME}(n) \subseteq \mathsf{TIMESPACE}(n^c, n^{o(1)})$ for some $c > (\sqrt{5}+1)/2$, for example $c = 2$.

Lecture 9

Interactive Proofs and Arthur-Merlin Games

April 27, 2004
Lecturer: Paul Beame
Notes: Chris Ré

9.1 Background and Motivation

In the 1980's two notions interactive computation were developed. One, due to Babai, originated in generalizations of NP to allow more powerful verifiers that include probabilistic verification. The other, due to Goldwasser, Micali, and Rackoff, originated in cryptography and was a means to the end of defining zero-knowledge proofs, protocols that allow a party in a cryptographic protocol to convince another party of some property without revealing additional information. (In this course we will not discuss the zero-knowledge aspects, however.) Today's lecture will focus on showing the relationship between these two definitions of interactive computation. (Combined, these two papers won the Gödel Prize.) The definitions here began research on the path to the PCP theorem.

9.2 Interactive Proof Preliminaries

We can view a typical NP algorithm as an interaction between an all-powerful *prover* P and a deterministic polynomial-time bounded *verifier* V. On a shared input x that the verifier wishes to prove is in L, the prover produces y, a certificate depending on x, sends the certificate y to the verifier and the verifier accepts if and only if $V(x, y) = 1$. This can be thought of as one round of interaction. The exact power of the prover is not important here but everything still must be verifiable by the deterministic polynomial time machine. So perhaps the amount of interaction is important? What if we were to allow more rounds of interaction as in the following figure?

Formally, we say that a *round* of an interactive protocol corresponds an uninterrupted sequence of communications of a single party. In this picture $y_1, y_2, \ldots, y_\ell$ denote the messages sent by the prover and $z_1, z_2, \ldots, z_{\ell-1}$ denote the messages sent in response by the verifier. We can formally define the verifier as an algorithm V and the actions of the verifier are given by by $z_1 = V(x_1, y_1)$, ..., $z_\ell = V(x, y_1, z_1, y_2, z_2, \ldots, y_\ell)$. The prover's actions can be defined similarly. The (still deterministic) polynomial time verifer accepts iff $V(x, y_1, z_1, y_2, z_2, \ldots, y_\ell) = 1$ at the end of the computation. (This is $2\ell - 1$ round computation.)

Definition 9.1. Given a Prover P and a Verifier V let $(P, V)(x)$ denote the output of the verifier on input x when the protocol (P, V) is executed.

Definition 9.2. (P, V) is an interactive proof for L if and only if

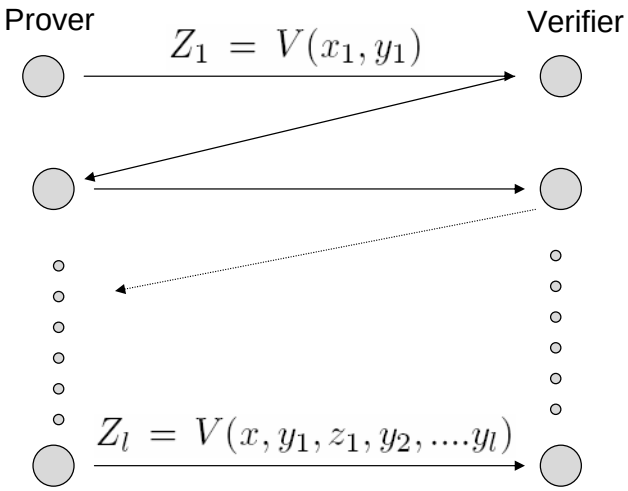

Figure 9.1: Prover Verifier Interaction

$$\begin{cases} (P,V)(x) = 1 & \text{for all } x \in L \\ (P^*,V)(x) = 1 & \text{for all } x \notin L \text{ for all provers } P^*. \end{cases}$$

We claim this is the same power as NP.

Lemma 9.1. *Interactive proofs with deterministic polynomial-time verifiers yields proofs for precisely the languages in* NP.

Proof. To see this, notice that for a multi-round proof the prover can determine in advance what the verifier will say at each response and simply send the entire sequence $(y_1, z_1, \ldots, y_\ell)$ to the verifier which yields a single round of interaction and thus NP. □

In each of the protocols it sufficed for the prover to have the power of NP in order to execute the protocol. However, our limits on the power of the verifier were what restricted the set of languages for which we had interactive proofs.

9.3 Interactive Proofs

Straight interaction with a deterministic verifier did not buy us any extra power. So instead we allow the Verifier to have access to a private random string r. Thus we can define a protocol pair $(P, V(r)$ and its actions on an input x as before. From now on we will assume that the verifier $V(r)$ runs in polynomial time as a function of the length of its inputs and that all messages are polynomial-length as a function of $|x|$.

Definition 9.3. A Prover P and a randomized verifier V with access to a random string r accepts a language L if and only if for some $\epsilon < 1/2$,

$$\begin{cases} \Pr_{r \in \{0,1\}^{|x|^{O(1)}}}[(P, V(r))(x) = 1] > 1 - \epsilon & \text{for } x \in L \\ \forall P^*.\ \Pr_{r \in \{0,1\}^{|x|^{O(1)}}}[(P^*, V(r))(x) = 1] \leq \epsilon & \text{for } x \notin L. \end{cases}$$

Notice that this is a BPP-like acceptance. It is common to say that the verifier is convinced if it accepts the interactive computation. A parameter we will be concerned with is the number of rounds. Notice that these are not round-trips but sending in one direction. This method of counting rounds corresponds nicely to alternation in the polynomial hierarchy. We can now define a complexity class of these interactions.

Definition 9.4. Define $\mathsf{IP}[k(n)]$ to be the set of languages L such that given an $x \in L$ there is a (polynomial-time randomized) protocol that takes at most $k(n)$ rounds to convince the verifier of x's membership in L.

Definition 9.5. $\mathsf{IP} = IP[n^{O(1)}] = IP[Poly(n)]$.

The following is an easy exercise.

Exercise 9.1. $\mathsf{IP} \subseteq \mathsf{PSPACE}$.

We will later prove that the reverse inclusion also holds, i.e. $\mathsf{IP} = \mathsf{PSPACE}$.

9.3.1 Graph Non-isomorphism $\in \mathsf{IP}[2]$

Definition 9.6. GRAPH-NON-ISOMORPHISM =
$\{\langle G_0, G_1\rangle \mid G_0, G_1$ are encodings of graphs and $\forall \sigma \in S_{|V(G_0)|},\ \sigma(G_0) \neq G_1\}$.

Notice the problem GRAPH-ISOMORPHISM of graph isomorphism is in NP since we can guess which permutation to use. This problem will be particularly interesting. Later we will show that if GRAPH-ISOMORPHISM were NP-complete then the polynomial-time hierarchy collapses.

Protocol

We will now give a 2 round protocol to decide GRAPH-NON-ISOMORPHISM. Both prover and verifier have access to G_0 and G_1.

$V \to P$: Verifier chooses $c \in_R \{0,1\}$, chooses $\sigma \in_R S_n$ where $n = |V(G_0)|$.
Verifier sends $\sigma(G_c)$ to the prover.

$P \to V$: The prover, with all possible computation power, determines to which of the two graphs this one is supposed to be isomorphic, say G_b for $b \in \{0,1\}$. (If the input graphs are isomorphic to each other the prover can choose b randomly.) The prover send b to the verifer.

Verification: The Verifier accepts iff $b = c$.

Discussion of Protocol

The important thing to notice is that if $\langle G_0, G_1\rangle$ is not in L, that is they are isomorphic graphs then the prover has exactly a 50-50 chance of guessing which value the verifier chose for c. If $\langle G_0, G_1\rangle \in L$ then the prover should never get it wrong.

This acceptance condition yields a probability only of $1/2$. We can massage this into the correct form that we can execute many copies in parallel. That is in the first round the verifier ships $G^1, .., G^k$ to the prover where each G^i is an independent uniformly random selection according to the first round of the

original protocol. The verifier accepts only if the prover answers correctly for each of the graphs. This does not increase the number of rounds but the prover only fools the verifier with probability $\frac{1}{2^k}$. Notice that it is crucial that the verifier not have access to the coin flips.

BPP and RP have *Amplification Lemmas* and these essentially carry over with us now using parallel independent invocations of the verifier as an analog of multiple independent copies of the machine.

Lemma 9.2 (IP Amplification Lemma). *Let $p(n)$ be a polynomial. Let V be a Verifier which on inputs of length n, a total of $g(n)$ messages each of length $m(n)$, using $\ell(n)$ random bits and error probability at most $1/2 - \delta(n)$. Then there is a V' such that $L(V) = L(V')$, using at most $g(n)$ messages, each of length $O(p(n)m(n)/\delta^2(n))$ using $O(p(n)\ell(n)/\delta^2(n))$ random bits and with an error probability at most $2^{-p(n)}$.*

Proof. (P', V') perform $O(p(n)/\delta^2(n))$ independent parallel simulations of (P, V) and V' takes the majority of the answers. Clearly this blows up the message by the corresponding orders of magnitude and the number of rounds is unchanged. The calculation for the probability of success is identitical that of Lemma 3.3 □

9.4 Arthur-Merlin Games

In this variant of interactive protocols at any point the in the protocol the Prover (now the all-powerful wizard Merlin) is allowed access to all the random coins used by the Verifier (now the not-so smart king Arthur) so far. With this definition it suffices to assume that the strings that are passed back and forth now simply contain the random guesses used by Arthur since it is clear that Merlin, who is all powerful, could easily have computed whatever string Arthur would have computed based on these random guesses.

The key is that Merlin is unaware of the outcomes of the coin before the Arthur sends them to him Notice that Merlin is powerful enough to simulate all possible flips of the coin ahead of time and therefore can play optimally. The game ends with a polynomial time deterministic verification based on the input and the message exchanged. We now use r_i to denote the i-th random string that Arthur sent. The acceptance condition is BPP-like as with ordinary interactive proofs. More precisely, there is a deterministic polynomial time verifier A and a prover M such that if $(r_1, y_1, \ldots, y_\ell)$ is the sequence of communicated values on input x, where each r_i is chosen uniformly at random, then $\Pr_{r \in \{0,1\}^{|x|^{O(1)}}}[A(x, z_1, r_1, y_1, ..., y_k) = 1] > 1 - \epsilon$ if $x \in L$ and for all choices of $(y_1^*, \ldots, y_k^*)$, $\Pr_{r \in \{0,1\}^{|x|^{O(1)}}}[A(x, z_1, r_1, y_1, ..., y_k) = 1] < \epsilon$ if $x \notin L$. We still count rounds in the half round trips and either party is allowed to go first.

9.4.1 Arthur-Merlin Complexity Classes

We denote the complexity class of languages accepted by Arthur-Merlin protocols by an alternating sequence of letters A and M where the number of letters equal to the number of rounds. For example, AMA = { L | there is a 3 round Arthur-Merlin protocol for L in which Arthur starts}.

Notice that ending at an A is like having a BPP machine to do the verification. The class M is NP and the class A is BPP.

Consider the following view of AM:

$L \in$ AM if and only if there is a polynomial-time verifier V, and a polynomial p such that $\forall x$,

$$\Pr[\ \text{Я}^{p(|x|)} r \exists^{p(|x|)} y V(x, y, r) = 1] \text{ is } \begin{cases} > 1 - \epsilon & \text{if } x \in L \\ \leq \epsilon & \text{if } x \notin L \end{cases}$$

The first quantifier acts like a BPP machine that makes one call to an NP oracle accepting its answer; so $\mathsf{AM} = \mathsf{BP} \cdot \mathsf{NP}$. The *value* of the Arthur-Merlin game is defined to be the probability that Arthur (the verifier) is convinced given optimal play by Merlin. We can express the value of the Arthur-Merlin game on input x using a form of quantifier, namely A as an Averaging quantifier and M as a Maximization quantifier. The value of the above AM protocol is then $A^{p(|x|)}r\ M^{p(|x|)}y.\ V(x, y, r)$. This use of A and M as the quantifiers is the source of the names Arthur and Merlin.

Now consider MA. In this case after doing all the nondeterministic work, we get access to a BPP machine, so we get $\mathsf{MA} = \mathsf{N} \cdot \mathsf{BPP}$. In quantifiers with a similar verifier the game would have a value $M^{p(|x|)}r\ A^{p(x)}y.\ V(x, y, r)$.

Definition 9.7. Define
$\mathsf{AM}[k] = \{L \mid L$ has a k round Arthur-Merlin game with Arthur starting$\}$ and
$\mathsf{MA}[k] = \{L \mid L$ has a k round Arthur-Merlin game with Merlin starting$\}$.

Today we will describe the surprising result that Arthur-Merlin games and interactive proofs are essentially equivalent in power.

Theorem 9.3 (Goldwasser, Sipser). *IP[t(n)] $\subseteq$ AM[t(n)+2]*

Next class we will prove the following Theorem.

Theorem 9.4 (Babai, Babai-Moran). *For constant $k \geq 2$, $\mathsf{AM} = \mathsf{AM}[k] = \mathsf{MA}[k+1]$, moreover, for any $t(n)$. AM[2t(n)] $\subseteq$ AM[t(n)+1].*

From these two theorems and the protocol for GRAPH-NON-ISOMORPHISM we derive.

Corollary 9.5. *GRAPH-NON-ISOMORPHISM* $\in \mathsf{AM}$

This is a bit surprising since our protocol relied so heavily on the secret coins which have disappeared in the Arthur-Merlin model.

Proof of Theorem 9.3. Suppose there is an interactive proof (P, V) for L. Merlin will convince Arthur that P *would have convinced* V for a large fraction of random strings r. The key to doing this will be to derive a short protocol that will convince Arthur that certain sets are large. This will be done using universal hashing.

Definition 9.8. A *(pairwise-independent) universal hash function family* is a family of functions, H, from a set U to a set V satisfying the following properties:

- For all $h \in H$, $h : U \to V$.
- For all $x \in U$ and for all $y \in V$, $\Pr_{h \in_R H}[h(x) = y] = \frac{1}{|V|}$.
- For all $x_1, x_2 \in U$ with $x_1 \neq x_2$ and for all $y_1, y_2 \in V$, $\Pr_{h \in_R H}[h(x_1) = y_1 \text{ and } h(x_2) = y_2] = \frac{1}{|V|^2}$.

Notice that the third property is a pairwise independence property, saying that knowing where one item is hashed does not give any information about where another item is hashed. The following is an example of such a family of hash functions.

Example 9.1. Let $U = \{0,1\}^n = \mathbb{F}_2^n$, $V = \{0,1\}^m = \mathbb{F}_2^m$. Choose an $n \times m$ matrix $A \in \{0,1\}^{n\times m} = \mathbb{F}_2^{n\times m}$, and a vector $v \in \{0,1\}^m = \mathbb{F}_2^m$. Then define $h_{A,v}(x) = Ax + v$ over $\mathbb{F}_2$. Let H be the set of all such functions. Note the similarity to the functions used in the proof of the Valiant-Vazirani lemma. The addition of the random vector v allows us to get around the fact that linear transformations always map the vector $0 \in V$ to $0 \in V$ and ensure that for h chosen at random, the output is a random vector in V and thus the second condition holds. The third condition holds by the same reasoning as in the proof of the Valiant-Vazirani lemma.

The following lemma gives the basis for the set size lower bound protocol.

Lemma 9.6. *Let $S \subseteq U = \{0,1\}^n$, V and a universal hash function family H from U to V. Select $t = 2n$ hash functions $h_1, \ldots, h_t \in_R H$ and $s = 3n$ points $r_1, \ldots, r_s \in_R V$.*

A. If $|S| \geq |V|/2$ then $\Pr[\exists i,\ j$ such that $r_j \in h_i(S)] \geq 1 - 2^{-n}$.

B. If $|S| < |V|/d$ then $\Pr[\exists i,\ j$ such that $r_j \in h_i(S)] < 6n^2/d$.

Proof. Fix $r \in V$ and $i \in \{1, \ldots, t\}$. For any $z \in U$, $\Pr_{h\in_R H}[h(z) = r] = \frac{1}{|V|}$ by definition of H. If $|S| \geq |V|/2$, let S' consist of the first $|V|/2$ elements of S. By inclusion-exclusion, for $h \in_R H$,

$$\begin{aligned}
\Pr[\exists z \in S.\ h(z) = r] &\geq \Pr[\exists z \in S'.\ h(z) = r] \\
&\geq \sum_{z\in S'} \Pr[h(z) = r] - \sum_{z'\neq z\in S'} \Pr[h(z) = r \text{ and } h(z') = r] \\
&= \left(\frac{|S'|}{|V|} - \frac{|S'|(|S'|-1)}{2}\right) \cdot \frac{1}{|V|^2} \\
&\geq 1/2 - 1/8 = 3/8
\end{aligned}$$

This implies that $E[|h(S')|] = \sum_{r\in V} \Pr[r \in h(S')] \geq 3|V|/8$. Now consider the probability that $\Pr_{h\in_R H}[|h(S')|] \geq |V|/4$? Using the fact that for any h, $|h(S')| \leq |S'| \leq |V|/2$, a standard Markov's inequality argument shows that this probability is at least $1/2$: Suppose the probability is $< 1/2$. Then $E[|h(S')|] < \frac{1}{2} \cdot |V|/2 + \frac{1}{2}|V|/4 = 3|V|/8$, contradicting our lower bound on $E[|h(S')|]$.

Therefore if we choose $h_1, \ldots, h_{n+1} \in_R H$,

$$\Pr_{h_1,\ldots h_{n+1}\in_R H}[\exists i.\ |h_i(S')| \geq |V|/4] \geq 1 - 2^{-2n}.$$

Suppose now that this holds. Thus the probability that every $r_j \in_R V$ is $\notin \bigcup_i h_i(S')$ is at most $3/4$ and the choices are independent. Therefore the probability that none of the r_j is in $\bigcup_i h_i(S)$ is at most $(3/4)^s < 2^{-n-1}$ for $s = 3n$. Thus the total failure probability is at most 2^{-n}.

Now suppose that $|S| \leq |V|/d$. Then $|\bigcup_{i=1} t h_i(S)| \leq t|S|$ and thus

$$\Pr[\exists i,\ j,\ \text{such that } r_j \in h_i(S)] \leq st\frac{|S|}{|V|} \leq 6n^2/d$$

as required. □

Thus, for Merlin to prove to Arthur that a set $S \in \{0,1\}^n$ is large relative to some bound 2^b, we have the following protocol:

Merlin sends to Arthur The value of the bound b.

Arthur sends to Merlin A sequence of random bits interpreted as $h_1, \ldots, h_{2n}$ independently chosen hash functions from $\{0,1\}^n$ to $V = \{0,1\}^b$ and $r_1, \ldots, r_{3n}$ independently chosen values from $V = \{0,1\}^b$.

Merlin send to Arthur A witness $z \in S$ such that $h_i(z) = r_j$ for some i and j.

If Merlin is unable to produce such a witness then the protocol fails. If $|S| \geq 2^b$ then Merlin is able to succeed with probability greater than $1 - 2^{-n}$; if $|S|$ is much smaller than 2^b then the protocol will only have a polynomially small success probability. It is important that Arthur can play this protocol even if the set S is implicit or known only to Merlin. The only difference is that if the set S is too small then Merlin is likely forced to produce a witness $z \in \{0,1\}^n - S$. Later Arthur may catch Merlin because of this. We will use this property in sequel.

Note. Note that we could have refined the above argument somewhat by taking advantage of the fact that all we need is $\bigcup_i h_i(S)$ to be large rather than any individual $h_i(S)$. A small modification of the above argument will show that

$$E[|h_i(S') - \bigcup_{i'<i} h_{i'}(S')|] \geq 3|V - \bigcup_{i'<i} h_{i'}(S')|/8.$$

Since the maximum value this quantity can take is at most $|V - \bigcup_{i'<i} h_{i'}(S')|$, the probability that it is at least $1/4$ of this maximum value is at least $1/6$ and these events are mutually independent. Thus a constant fraction of the time each additional hash function reduces the size of the uncovered portion of V by a constant factor. By choosing $O(n)$ independent hash functions (or even $O(b)$ if a failure probability only at most 2^{-b} is required) we can assure almost certainly that every point of V is covered by the image of S' under some hash function. Thus, only a single random query r is required.

Protocol Overview By the amplication lemma we can assume that that $(P, V(r))(x)$ accepts with very high probability if $x \in L$ and any $(P^*, V(r))(x)$ accepts with at most exponentially small probability.

Suppose that the set of all random strings used by the verifier V is chosen from $\{0,1\}^\ell$. Merlin contends that the set of random strings for which V would have been convinced is not very small, say at least $2^{\ell-1}$. Let $A_\epsilon = \{r \mid (P, V(r)) \text{ accepts}\}$ be this set of random strings. We need to decide whether or $|A_\epsilon|$ is nearly 2^ℓ or exponentially smaller than 2^ℓ.

We first will assume that IP protocol is a 2-round protocol of the style that was used for GRAPH-NON-ISOMORPHISM in which the verifier V first sends a string z and then the prover P sends a string y to the verifier who makes a final decision based also the random string.

Let $z = V(x, \epsilon, r)$, the string that V would have been sent to P given the random string $r \in \{0,1\}^\ell$. Associated with each message z that V sends in this round there is an optimal response string $y = y(z)$ that maximizes the probability that $V(x, (z, y), r)$ accepts. (Assume without loss of generality that $z \in \{0,1\}^m$.) Now define the set

$$A_z = \{r \mid z = V(x, \epsilon, r) \text{ and } V(x, (z, y(z)), r) \text{ accepts } \},$$

the set of all random strings r such that the verifier initially sent z, the prover can respond with $y(z)$ and the verifier will accept. Clearly $A_\epsilon = \bigcup_z A_z$ where the union is disjoint.

The goal is to give a protocol for an AM protocol that allows Arthur to be convinced that the Verifier would have accepted this input under the AM acceptance condition. Merlin will do this by proving to Arthur

that with large probability there is a large set of z such that A_z is large. To do this, Merlin will choose a set S_1 of z's each of which has roughly the same size.

The Set S_1 a: Command not found. where we have chosen b_1 to maximize $|\bigcup_{z \in S_1} A_z|$. That is, we put the A_z into buckets based on the order of magnitude of their size; after binning, we choose the bin that contributes the most to A_ϵ. By construction,

- $\sum z \in S_1 |A_z| \geq \frac{|A_\epsilon|}{\ell}$ and
- for each $z \in S_1$, $2^{b_1 - 1} < |A_z| \leq 2^{b_1}$.

Merlin will pick the set S_1 and send the value $s_1 = \lceil \log_2 |S_1| \rceil$ to Arthur and convince Arthur that S_1 is large relative to 2^{s_1}. In doing this using the set size lower bound protocol, Merlin will identify and element $z \in S_1$. He will then prove that A_z is large relative to 2^{b_1}. If $b_1 + s_1$ is large enough, Arthur will accept.

These will be done by applications of the hashing protocol above. As a result, Arthur can be convinced of the truth of the assertion if we show him that the set is at least a constant fraction $\frac{2^\ell}{\ell}$.

The Protocol

- $M \rightarrow A$: (Round 0)

  Merlin computes the set S_1, the value b_1 and sends $s_1 = \lceil \log_2 |S_1| \rceil$ and b_1 to Arthur.

- $A \rightarrow M$: (Round 1)

  Arthur sends 2ℓ random hash functions $h_1, \ldots, h_{2\ell}$ from $\{0,1\}^m$ to $\{0,1\}^{s_1}$ and 3ℓ random challenge strings $r_1, \ldots, r_{3\ell} \in_R \{0,1\}^{s_1}$ to Merlin.

  Arthur sends 2ℓ random hash functions $h'_1, \ldots, h'_{2\ell}$ from $\{0,1\}^\ell$ to $\{0,1\}^{b_1}$ and 3ℓ random challenge strings $r'_1, \ldots, r'_{3\ell} \in_R \{0,1\}^{b_1}$ to Merlin.

- $M \rightarrow A$: (Round 2)

  Merlin produces a string $z \in S_1$ such that $h_i(z) = r_j$ for some i and j if possible. (Otherwise Merlin chooses an arbitrary string z that maps to one of the r_j's if possible.)

  Merlin sends $(z, y(z))$ as well as (i, j) to Arthur.

  Merlin produces a string $r \in A_z$ such that $h'_{i'}(r) = r'_{j'}$ for some i' and j' if possible. (Otherwise, Merlin chooses an arbitrary string r that maps to one of the r_j's if possible.)

  Merlin sends r as well as (i', j') to Arthur.

- Verify: In the final deterministic verification Arthur checks that $h_i(z) = r_j$, that $h'_{i'}(r) = r'_{j'}$, that $V(x, \epsilon, r) = z$, that $V(x, (z, y), r)$ accepts (i.e., that $r \in A_z$), and that $s_1 + b_1 \geq \ell - \log_2 \ell - 1$.

Now let us verify that this yields a good Arthur-Merlin protocol.

If $\Pr_r[(P, V(r))(x) = 1] > 1/2$: In this case, $|A_\epsilon| > 2^{\ell-1}$ and thus

$$\sum_{z \in S_1} |A_z| \geq |A_\epsilon|/\ell > 2^{\ell-1}/\ell \geq 2^{\ell - \log_2 \ell - 1}.$$

Since $|S_1| \geq 2^{s_1-1}$, by Lemma 9.6, Merlin will be able to find the required $z \in S_1$ with failure probability at most $2^{-\ell}$. Then for this z, $|A_z| \geq 2^{b_1-1}$ so again by Lemma 9.6 and the definition of A_z, Merlin will be able to find an r such that h' hashes r correctly, $V(x, \epsilon, r) = z$, and $V(x, (z, y), r)$ accepts. Finally, observe that

$$\sum_{z \in S_1} |A_z| \leq \sum_{z \in S_1} 2^{b_1} = |S_1| \cdot 2^{b_1} \leq 2^{b_1 + s_1}$$

and thus $b_1 + s_1 \geq \ell - \log_2 \ell - 1$.

If $\Pr_r[(P, V(r))(x) = 1] < 1/\ell^7$: In this case $|A_\epsilon| \leq 2^\ell/\ell^7$. Suppose that Merlin tries to cheat Arthur. What is the chance that Arthur will be convinced? In order for Arthur to be convinced Merlin must send b_1 and s_1 in Round 0 such that $b_1 + s_1 \geq \ell - \log_2 \ell - 1$, that is, $2^{b_1+s_1} \geq \frac{2^\ell}{2\ell}$. Choose $d = \ell^3$. Let S_1' be the set of all z such that $|A_z| \geq 2^{b_1}/d$. The size of S_1' is at most $d \cdot |A_\epsilon|/2^{b_1} \leq 2^{\ell - b_1}/\ell^4 \leq 2^{s_1 + \log_2 \ell + 1}/\ell^4 = 2^{s_1+1}/d$. The probability that Merlin could produce an element z of S_1' in response to the hash challenge for S_1 is $O(\ell^2/d)$ which is $O(1/\ell)$. If Merlin does not produce such an element then the probability that Merlin is able to produce an element r for A_z that will convince Arthur is also only $O(1/\ell)$. Therefore, the total probability that Arthur is convinced in this case is $O(1/\ell)$.

This proves correctness in the two round case.

More Rounds In general, for a procotol with more rounds, note that for any prefix of the computation, once r is fixed, whether or not V will accept beginning with that prefix is a deterministic property using the optimal play of Merlin. Therefore once Merlin fixes $(z_1, y_1(z_1))$ for his first response, we can define sets

$$A_{z_1,z} = \{r \mid V(x, \epsilon, r) = z_1 \text{ and } V(x, (z_1, y_1(z_1)), r) = z \text{ and } V(x, (z_1, y_1(z_1), z, y_2(z), \ldots), r) \text{ accepts } \},$$

$$A_{z_1,z_2,z} = \{r \mid s = (z_1, y_1(z_1), z_2, y_2(z_2), z, y_3(z)) \text{ is a valid execution on input } x \text{ and } V(x, (s \ldots), r) \text{ accepts } \},$$

etc. At each round, Merlin will choose an S_i consisting of those z in the highest weight bin for the sets $A_{z_1,\ldots,z_{i-1},z}$. At each round, Merlin will send Arthur $s_i = \lceil \log_2 |S_i| \rceil$ and convince Arthur that $|S_i| \geq 2^{s_i - 1}$ by sending z_i and $y_i(z_i)$. For the last round of the prover's communication, Merlin will send b_k in addition to s_k, and convince Arthur that $|A_{z_1,\ldots,z_k}| \geq 2^{b_k}$ by sending a choice of r. If there are $2k$ rounds beginning with then Arthur then there will be a loss of ℓ^k in the size of $|A_\epsilon|$ at most so Arthur will accept if all the hash conditions have been met, the final random string and the sequence of z_i and y_i will yield acceptance, and $b_k + \sum_{i=1}^k s_i \geq \ell - k \log_2 \ell - 1$. By a similar argument to the 2 round case, if the original acceptance probability is at most $1/\ell^{8k-1}$, then Arthur is convinced only an $O(k/\ell)$ fraction of the time.

Number of Rounds Notice how many rounds it took to prove the claim. In the new protocol we have added one communication from Merlin to Arthur of the set size s_1 (and b_1 in the two round case) at the beginning and one communication at the end for Merlin to send Arthur the random string r in response to the last challenge. If the protocol already began and ended with Arthur then this adds two rounds. If the protocol already began or ended with Merlin (as in the 2 round case) then we add fewer rounds.

□

Lecture 10

Arthur-Merlin Games

April 29, 2004
Lecturer: Paul Beame
Notes: Ethan Phelps-Goodman

10.1 The Collapse Lemma

The main lemma of this lecture is the following:

Lemma 10.1 (Babai). $\mathsf{MA} \subseteq \mathsf{AM} = \mathsf{AM}[k] = \mathsf{MA}[k+1]$, *for any constant* k. *That is,* AM *can simulate any constant round Arthur-Merlin game.*

Proof. Suppose that there are two rounds of the Arthur-Merlin game that go in sequence MA.... We show how to simulate them by a modified game that begins AM... and has the same alternation pattern after the first two rounds. This will be enough to prove the claim because it will also allow us to convert any game sequence AMA... to AAM... $= \mathsf{AM}$... and MAM... to AMM... $= \mathsf{AM}$....

It is somewhat intuitive that AM should be more powerful than MA, because in the former Merlin gets to look at the random bits before deciding on his answer. The argument first shows that one can bound the increase in Merlin's power and then use amplification of the original protocol to ensure that this increase in Merlin's convincing power is not enough to allow him to cheat.

Start with a zero-one random variable $V(x, y, r)$. Think of V as determining whether the protocol accepts in the remaining rounds. Define $H(x, yr) = \Pr[V(x, y, r) = 1]$, where the probability is over remaining rounds of the protocol (not r). Let $\Psi(x) = MyAr\ H(x, y, r)$ be the probability that theMA... protocol accepts x. Also, say that the ys are taken from the set Y and the rs are taken from the set R. We use A for averaging over the set R and M for maximizing over the set Y.

Lemma 10.2. *For any function* H, $ArMy\ H(x, y, r) \leq |Y| MyAr\ H(x, y, r)$.

This quantifier swap will be the basis for simulating the MA... protocol by an AM... protocol. The size of Y will be exponential in the length of x, so we lose a lot of accuracy by switching from MA to AM, but we will show that this loss can be counteracted by amplifying the success probability of the original protocol sufficiently using amplification for Arthur-Merlin games which is essentially a variant of the amplification from Lemma 9.2.

Proof. We can get a crude upper bound on the maximum y by summing over all choices of y:

$$ArMy\ H(x, y, r) \leq Ar\Sigma_{y \in Y}\ H(x, y, r)$$

The averaging quantifier is really just a summation divided by the number of terms in the summation, so it commutes with the sum over y:

$$ArMy\ H(x,y,r) \leq \Sigma_{y\in Y} Ar\ H(x,y,r)$$

Next we upper bound the sum over y by the maximum y times the number of choices of y:

$$ArMy\ H(x,y,r) \leq |Y| MyAr\ H(x,y,r)$$

This gives the desired bound. □

Back to the proof of the main lemma, we started with an MA... protocol, which looks like

$$\exists y \text{Я} r\ V(x,y,r) = 1$$

Our new protocol will start by picking a sequence of random strings, $r_1 \ldots r_m \in R$, where m is polynomial in the length of x. Our new AM protocol will look like

$$\text{Я} r_1 \ldots r_m My\ Majority_{i=1}^{m}(V(x,y,r_i)).$$

Lemma 10.3. *The acceptance probability of the new* AM*... majority protocol is*

$$1 - 2^m(1-\Psi(x))^{m/2} \leq A(r_1 \ldots r_m) My\ Majority_i(V(x,y,r_i)) \leq 2^m |Y| \Psi(x)^{m/2}$$

Proof. We will prove the lower bound first. For the protocol to reject, at least half of the r_is must lead to a rejection. This give us

$$\exists I \subseteq \{1 \ldots m\},\ |I| = \lceil m/2 \rceil \text{ such that } V(x,y,r_i) \text{ rejects for all } i \in I.$$

In the new AM protocol Merlin could send the same string as in the MA protocol irrespective of the random string r_i. This would give the same success probability $\Psi(x)$, which we will use as a lower bound for the success of one invocation of $V(x,y,r_i)$. Then we have

$$\begin{aligned} \Pr[\text{all trials in } I \text{ fail}] &\leq (1-\Psi(x))^{|I|} \\ &= (1-\Psi(x))^{m/2} \end{aligned}$$

We can upper bound the number of choices of I by 2^m, so

$$\text{total failure probability } \leq 2^m(1-\Psi(x))^{m/2}$$

This gives the lower bound on success claimed in the lemma.

For the upper bound we use the same definition of I, but want all trials in I to accept. This happens with probability:

$$\begin{aligned} \Pr[\forall i \in I.\, V(x,y,r) = 1] &= \Pi_{i\in I} \Pr[V(x,y,r) = 1] \\ &= \Pi_{i\in I} H(x,y,r) \end{aligned}$$

For any fixed y, we can average over the choices of $r_1 \dots r_m$:

$$\begin{aligned} A(r_1 \dots r_m)\ Majority_i(V(x,y,r_i)) &\leq \frac{\Sigma_{r_1 \dots r_m \in R^m} \Sigma_{I \subseteq 1 \dots m, |I| = \lceil m/2 \rceil} \Pi_{i \in I} H(x,y,r_i)}{|R|^m} \\ &\leq \Sigma_I \frac{\Sigma_{r_1 \dots r_m \in R^m} \Pi_{i \in I} H(x,y,r_i)}{|R|^m} \end{aligned}$$

We can replace the average over elements of R^m with an average over elements of R^I since only indices from I affect the probability, so:

$$\begin{aligned} A(r_1 \dots r_m) Majority_i(V(x,y,r_i)) &\leq \Sigma_I \frac{\Sigma_{\vec{r} \in R^I} \Pi_{i \in I} H(x,y,r_i)}{|R|^{|I|}} \\ &= \Sigma_I \Pi_{i \in I} \left(\Sigma_{r_i \in R} \frac{H(x,y,r_i)}{|R|} \right) \\ &= \Sigma_I \left(Ar\ H(x,y,r) \right)^{m/2} \end{aligned}$$

Now replace the arbitrary y with the best possible y:

$$\begin{aligned} MyA(r_1 \dots r_m)\ Majority_i(V(x,y,r_i)) &\leq \Sigma_I \left(MyArH(x,y,r) \right)^{m/2} \\ &= \Sigma_I (\Psi(x))^{m/2} \\ &\leq 2^m \Psi(x)^{m/2} \end{aligned}$$

Combining this with lemma 10.3 we get

$$A(r_1 \dots r_m) My\ Majority_i(V(x,y,r_i)) \leq 2^m |Y| \Psi(x)^{m/2}$$

□

Now that we have established the bounds we will set the value of the parameters. Let $m = 2 \log_2 |Y| + 4$. Assume without loss of generality that for $x \notin L$, $\Psi(x) \leq 1/8$ and for $x \in L$, $\Psi(x) \geq 7/8$. Then the probability of the new AM protocol accepting an $x \notin L$ is at most:

$$\begin{aligned} A(r_1 \dots r_m) My\ Majority_i(V(x,y,r_i)) &\leq 2^m |Y| \Psi(x)^{m/2} \\ &\leq 2^m 2^{-3m/2} |Y| \\ &= 2^{-\log_2 |Y| - 2} |Y| \\ &= \frac{1}{4} \end{aligned}$$

The probability of accepting an $x \in L$ is at least:

$$\begin{aligned} A(r_1 \dots r_m) My\ Majority_i(V(x,y,r_i)) &\geq 1 - 2^m(1-\Psi(x))^{m/2} \\ &\geq 1 - 2^m(\frac{1}{8})^{m/2} \\ &= 1 - 2^{-m/2} \\ &= 1 - 2^{-\log_2 |Y| - 2} \\ &= 1 - \frac{1}{4|Y|} \\ &\geq \frac{3}{4} \end{aligned}$$

This concludes the proof. □

10.2 Variable-round games

The proof in the last section allows any constant number of Arthur-Merlin rounds to be reduced to a single AM round. This can't be extended directly to variable round protocols because there is a polynomial blowup in each switch, which would lead to an exponential blowup in variable round protocols. The following lemma allows for variable round reductions. We will only outline the idea of the proof.

Lemma 10.4 (Babai-Moran). $\mathsf{AM}[2t(n)] \subseteq \mathsf{AM}[t(n)+1]$.

Proof Sketch. The main idea is to convert a single MAM round to an AMA round. As before, Arthur will start by sending a collection of random seeds $r_1 \dots r_m$. After Merlin gives a response $y_1 \dots y_m$ as well as $z_1, dotscz_m$, Merlin's responses for the third round of the game given the correspond choices of y_i and r_i. Arthur sends a random $i \in \{1 \dots m\}$. The protocol the proceeds as if the original game had been played with the sequence y_i, r_i, z_i had been the only interaction so far. This leaves no blowup in the number of games that must be continued (although there is a polynomial blow-up for this triple of rounds). This allows all the MAM seqeunces to be replaced by AMA sequences in parallel with a single polynomial size blow-up. The general idea is that if the overwhelming majority of continuations don't allow Merlin to cheat too much then Arthur will likely pick one on which he won't be fooled. □

10.3 AM games and the Polytime Hierarchy

We can now relate Arthur-Merlin games and the polynomial time hierarchy.

Lemma 10.5. $\mathsf{AM} \subseteq \Pi_2\mathsf{P}$

Proof. The proof is a direct analogue of the Sipser-Gacs-Lautemann proof of $\mathsf{BPP} \subseteq \Sigma_2\mathsf{P} \cap \Pi_2\mathsf{P}$ (see Lecture 3 Theorem 3.5). In that proof we defined $S = \{r \mid M(x,r) = 1\}$ to be the set of random strings that lead to acceptance of M on input x. We then gave a $\Sigma_2\mathsf{P}$ algorithm that accepts the input x if and only if the set S is large. Since BPP is closed under complement, by applying the protocol to $\bar{S}$ we obtain a $\Sigma_2\mathsf{P}$ algorithm for $\bar{A}$ and thus a $\Pi_2\mathsf{P}$ algorithm for L. In particular, this $\Pi_2\mathsf{P}$ expression says that $x \in A$ if and only if

$$\forall t_1 \dots t_{p(|x|)} \in \{0,1\}^{p(|x|)} \exists^{p(|x|)} r \text{ for all } j \in \{1, \dots, p(|x|)\}.\ M(x, r \oplus t_j) = 0.$$

(Since there are only $p(|x|)$ values of j the inner portion can be recognized in polynomial time.)

For the current proof we view AM as BP $\cdot$ NP, and the set S associated with a language L defined by an AM protocol with verifier V is $S = \{r \mid \exists y\, V(x, y, r) = 1\}$. The new expression is then

$$\forall t_1 \dots t_{p(|x|)} \in \{0,1\}^{p(|x|)} \exists^{p(|x|)} r \text{ such that for all } j \in \{1, \dots, p(|x|)\},\ \exists y\, V(x, y, r \oplus t_j) = 0,$$

which is equivalent to

$$\forall t_1 \dots t_{p(|x|)} \in \{0,1\}^{p(|x|)} \exists^{p(|x|)} r \exists (y_1, \dots, y_{p(|x|)}) \text{ s.t. for all } j \in \{1, \dots, p(|x|)\},\ V(x, y_j, r \oplus t_j) = 0.$$

This is a Π_2P expression for L. □

Lemma 10.6. MA $\subseteq \Sigma_2$P $\cap\, \Pi_2$P

Proof. We can view MA as N $\cdot$ BPP. We know that BPP has a Σ_2P representation. Adding on another existential quantifier for the Merlin step gives a language that is still in Σ_2P, so MA $\subseteq \Sigma_2$P. We know that MA $\subseteq$ AM, and that AM $\subseteq \Pi_2$P, so MA $\subseteq \Pi_2$P as well. □

10.4 Graph Isomorphism is unlikely to be NP-complete

With the results from today we can easily show that GRAPH-ISOMORPHISM being NP-complete leads to the polynomial time hierarchy collapsing.

Lemma 10.7. *If* coNP $\subseteq$ AM *then* PH $= \Sigma_2$P $\cap\, \Pi_2$P $=$ AM.

Proof. Let $L \in \Sigma_2$P. We will show that under the assumption coNP $\subseteq$ AM we get $L \in$ AM $\subseteq \Pi_2$P, which causes the hierarchy to collapse at the second level. Since $L \in \Sigma_2$P, by Theorem 2.2 we can express L as

$$L = \{x \mid \exists^{p(|x|)} y.\ (x, y) \in L_1\}$$

where $L_1 \in$ coNP$\}$. If coNP $\subseteq$ AM then $L_1 \in$ AM, so $L \in$ MAM by treating the existential quantifier as a Merlin step. But by the collapse lemma from this lecture, MAM $=$ AM $\subseteq \Pi_2$P, and the hierarchy collapses to Σ_2P $=$ AM $= \Pi_2$P. □

Corollary 10.8. *If GRAPH-ISOMORPHISM is* NP*-complete then* PH $= \Sigma_2$P $\cap\, \Pi_2$P $=$ AM.

Proof. If GRAPH-ISOMORPHISM is NP-complete then GRAPH-NONISOMORPHISM is coNP-complete. We know from last lecture that GRAPH-NONISOMORPHISM $\in$ AM, implying that coNP $\subseteq$ AM. From the above lemma this causes the polynomial time hierarchy to collapse as in the conclusion. □

Lecture 11

IP, PH, and PSPACE

May 4, 2004
Lecturer: Paul Beame
Notes: Daniel Lowd

11.1 IP and PH

Theorem 11.1 (Lund-Fortnow-Karloff-Nisan). *There is a polynomial length interactive proof for the predicate PERM$(A) = k$.*

Before we prove this theorem we note a number of corollaries.

Corollary 11.2. *There exist polynomial length interactive proofs for all of $\#\mathsf{P}$.*

Corollary 11.3. $\mathsf{PH} \subseteq \mathsf{IP}$

This is surprising, because a constant number of alternations is equivalent to two alternations, but an unbounded number yields PH. Later, we will prove that there exist interactive proofs for everything in PSPACE.

Proof of Corollaries. These follow from the easy observation that IP is closed under polynomial-time (Turing) reduction and by Toda's theorem. □

Remark. In the journal version of their paper, Lund, Fortnow, Karloff, and Nisan gave an alternative direct protocol proving Corollary 11.2. This proof is given in full in Sipser's text. We present the protocol for the permanent directly both because it is interesting in its own right and because the permanent problem motivates the whole approach for these proofs.

Proof of Theorem 11.1. We perform the computation of PERM over some finite field $\mathbb{F}$, where $|\mathbb{F}| > 10n^3$. We also assume that $\{1, \ldots, n\} \subset \mathbb{F}$, although we can remove this restriction by simply choosing n distinct field elements of $\mathbb{F}$ that can substitute for $\{1, \ldots, n\}$.

Recall the definition of PERM:

$$\text{PERM}(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} a_{i,\sigma(i)}$$

where $a_{i,j}$ is the element in the ith row and jth column of matrix A and S_n is the set of all permutations from $\{1, \ldots, n\}$ to itself.

Note that PERM is a multivariate polynomial in the inputs, with total degree n and degree at most 1 in each variable $a_{i,j}$. Furthermore, we can define PERM recursively via the following self-reduction:

Definition 11.1. Let $A(i|j)$ be the $(n-1) \times (n-1)$ matrix equal to A with its ith row and jth column removed.

Claim 2. $\text{PERM}(A) = \sum_{\ell=1}^{n} a_{1,\ell}\text{PERM}(A(1|\ell))$.

The proof for this reduction is by direct application of the definition:

$$\begin{aligned}\sum_{\ell=1}^{n} \sum_{\substack{\sigma \in S_n, \\ \sigma(1)=\ell}} \prod_{i=1}^{n} a_{i,\sigma(i)} &= \sum_{\ell=1}^{n} a_{1,\ell} \sum_{\sigma \in S_n} \prod_{i=2}^{n} a_{i,\sigma(i)} \\ &= \sum_{\ell=1}^{n} a_{1,\ell} \sum_{\sigma \in S_{n-1}} \prod_{i=1}^{n-1} a_{i',\sigma'(i')}\end{aligned}$$

where $i' = i+1$ and $\sigma'(i') = \begin{cases} \sigma(i') & \text{for } \sigma(i') < \ell \\ \sigma(i')+1 & \text{for } \sigma(i') \geq \ell \end{cases}$

$$= \sum_{\ell=1}^{n} a_{1,\ell} \cdot \text{PERM}(A(1|\ell)).$$

To prove that $\text{PERM}(A) = k$, it suffices to prove the values of $\text{PERM}(A(1|\ell))$ for $\ell = 1, \ldots n$. Of course, a fully recursive algorithm would yield $n!$ subproblems, saving us nothing over direct computation. Instead, the prover will give *one* proof that gives allows us to recursively prove values for all $\text{PERM}(A(1|\ell))$ via a single proof rather than n separate proofs.

To do this we will use a representation of these values of the permanent as polynomials. The following are the two basic properties of polynomials that we will use.

Proposition 11.4. *If $p \neq 0$ is a degree d univariate polynomial over $\mathbb{F}$ then p has at most d roots.*

Corollary 11.5. *For any degree d univariate polynomial $f \neq 0$ over $\mathbb{F}$, $Pr_{r \in_R \mathbb{F}}[f(r) = 0] \leq d/|\mathbb{F}|$.*

Proposition 11.6. *(Interpolation Lemma) Given any distinct set of points $\{b_1, \ldots b_n\} \subset \mathbb{F}$ and any (not necessarily distinct) $\{c_1, \ldots, c_n\} \subset \mathbb{F}$ there is a degree n univariate polynomial $p \in \mathbb{F}[x]$ such that $p(b_i) = c_i$ for $i = 1, \ldots, n$.*

Basic Idea Write $B(\ell) = A(1|\ell)$. Then

$$B(\ell) = \begin{bmatrix} b_{1,1}(\ell) & \cdots & b_{1,n-1}(\ell) \\ \vdots & \ddots & \vdots \\ b_{n-1,1}(\ell) & \cdots & b_{n-1,n-1}(\ell) \end{bmatrix}$$

Each $b_{i,j}(\ell)$ is a function: $\{1, \ldots, n\} \to \mathbb{F}$. By the interpolation lemma, there are degree n polynomials $p_{i,j}(z)$ such that $p_{i,j}(\ell) = b_{i,j}(\ell)$ for $\ell = 1, \ldots n$. Write $B(z)$ for this matrix of polynomials. Then $B(\ell) = A(1|\ell)$ for $\ell = 1, \ldots, n$ but is also defined for other values of z.

Given this matrix of polynomials $B(z)$,

$$\text{PERM}(A) = \sum_{\ell=1}^{n} a_{1,\ell}\text{PERM}(B(\ell))$$

Observe that:

A. $\text{PERM}(B(z))$ is a degree $(n-1)n$ univariate polynomial over $\mathbb{F}$.

B. Given A, both players can compute what $B(z)$ is.

Interactive protocol to prove that $\text{PERM}(A) = k$

- Prover computes $f(z) = \text{PERM}(B(z))$ and sends the $n(n-1)+1$ coefficients of f to the verifier.
- Verifier checks that $\sum_{\ell=1}^{n} a_{1,\ell} f(\ell) = k$. If good, chooses $r_1 \in_R \mathbb{F}$ and sends it to the prover.
- Prover continues with a proof that $\text{PERM}(B(r_1)) = f(r_1)$.

The proof continues until matrix size is one. In that case the Verifier computes the permanent directly by checking that the single entry of the matrix is equal to the claimed value for the permanent.

Note. Note that is very important in this protocol r_i could take on a value larger than n. In other words, $B(r_i)$ might not be a submatrix of A at all.

Clearly, if $\text{PERM}(A) = k$ then the prover can always convince the verifier.

Suppose that $\text{PERM}(A) \neq k$. At each round there is an $i \times i$ matrix A_i and an associated claimed value k_i for the permanent of A_i where $A_n = A$ and $k_n = k$. The Prover can cause the Verifier to accept only if $\text{PERM}(A_1) = k_1$. Therefore in this case there is some round i such that $\text{PERM}(A_i) \neq k_i$ but $\text{PERM}(A_{i-1}) = k_{i-1}$. Let $B(z)$ be the $(i-1) \times (i-1)$ polynomial matrix associated with A_i and $f(z)$ be the degree $i(i-1)$ polynomial sent by the Prover in this round. Either $f(z) = \text{PERM}(B(z))$ as polynomials or not.
If $f(z) = \text{PERM}(B(z))$, then $\sum_{\ell=1}^{n} a_{1,\ell} f(\ell) = \sum_{\ell=1}^{n} a_{1,\ell} \text{PERM}(B(\ell)) \neq k$, and the verifier will reject the proof immediately.
If $f(z) \neq \text{PERM}(B(z))$, then $f(z) - \text{PERM}B(z)) \neq 0$ and therefore $Pr_{r \in_R \mathbb{F}}[f(r) = \text{PERM}(B(r))] \leq i(i-1)/|\mathbb{F}|$. In other words, the probability that the prover can "fool" the verifier in this round is at most $i(i-1)/|\mathbb{F}|$. Therefore, the total probability that the Prover succeeds in convincing the Verifier of an incorrect value is at most $\sum_{i=2}^{n} i(i-1)/\mathbb{F} < n^3/|\mathbb{F}| \leq 1/10$ for $|\mathbb{F}| \geq 10n^3$. (In fact, the sum is at most $(n^3 - n)/(3||)$ so $|\mathbb{F}| \geq n^3$ suffices.) □

The above proof shows that there are interactive proofs for coNP. A constant number of rounds is unlikely unless the polynomial-time hierarchy collapses. However, in the above protocol the prover requires the ability to solve a #P-hard problem.

Open Problem 11.1. What prover power is required to prove coNP $\subseteq$ IP?

11.1.1 Low Degree Polynomial Extensions

A key idea of the above argument was to use *low degree polynomial extensions*. This involves taking a function $f : I \to \mathbb{F}$, in this case $I = \{1, \ldots, n\}$, extending it to a polynomial $P_f : \mathbb{F} \to \mathbb{F}$, and checking P_f on random points of $\mathbb{F}$.

To apply this we used the fact that the function in question we wished to compute could be expressed as a multivariate polynomial of low total degree.

11.2 IP equals PSPACE

In this section we prove the following characterization theorem for IP.

Theorem 11.7 (Shamir, Shen). IP = PSPACE

Proof. (Following Shen.) We will prove the hard direction, namely that PSPACE $\subseteq$ IP; the other direction is left as an exercise.

The key idea of this proof will also involve low degree polynomial extensions. In order to use this we need the following facts about finite fields.

A. For any integer n, there exists a prime p such that $n \leq p \leq 2n$.

B. For any prime p and integer $k \geq 0$, there exists a finite field $\mathbb{F}_{p^k}$ with p^k elements.

We construct an IP protocol for TQBF using low-degree polynomial extensions over a small finite field $\mathbb{F}$. Specifically, we can choose a small field $\mathbb{F}$ with $n^3 m \leq |\mathbb{F}| \leq 2n^3 m$, where m is the number of 3-CNF clauses and n is the number of variables in the TQBF formula $\Psi = \exists x_1 \forall x_2 \cdots Q_n x_n \psi(x_1, \ldots, x_n)$ where ψ is a 3-CNF formula.

11.2.1 Arithmetization of Boolean formulas

Create multivariate polynomial extensionds for Boolean formulas as follows:

$$\begin{aligned} f \wedge g &\mapsto P_f \cdot P_g \\ x_i &\mapsto x_i \\ \neg f &\mapsto (1 - P_f) \\ (f \vee g) = \neg(\neg f \wedge \neg g) &\mapsto 1 - (1 - P_f)(1 - P_g). \end{aligned}$$

We use the notation $P_f \circledast P_g$ as a shorthand for $1 - (1 - p_f)(1 - p_g)$. Applying these operations $P_\psi(x_1, \ldots, x_n)$ is of degree $\leq m$ in each variable, with a total degree of $\leq 3m$.

Continuing this in the obvious way we obtain that

$$P_{\forall x_n f}(x_1, \ldots, x_{n-1}) = P_f(x_1, \ldots, x_{n-1}, 0) \cdot P_f(x_1, \ldots, x_{n-1}, 1)$$

and

$$P_{\exists x_n f}(x_1, \ldots, x_{n-1}) = P_f(x_1, \ldots, x_{n-1}, 0) \circledast P_f(x_1, \ldots, x_{n-1}, 1).$$

We want to know if $P_\Psi() = 1$. The obvious analog to our proof for PERM has a problem: the degree of the polynomial doubles at each quantification step and thus the univariate polynomials we will create will have exponential degree. The solution rests on the fact that our polynomial need only be correct on inputs over $\{0, 1\}$, which yields only two points per variable. Thus a polynomial of linear degree in each variable will suffice. For this purpose we introduce a new degree reduction operation Rx_i.

Definition 11.2. $P_{Rx_i f}(x_1, \ldots, x_n) = x_i \cdot P_f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n) + (1 - x_i) \cdot P_f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)$

We now replace Ψ by the formal sequence

$$\Psi_0 = \exists x_1 R x_1 \forall x_2 R x_1 R x_2 \exists x_3 R x_1 R x_2 R x_3 \cdots Q_n x_n R x_1 R x_2 \cdots R x_n \psi(x_1, \ldots, x_n).$$

While the $\exists$ and $\forall$ operators increase the polynomial degree, the R operators bring it back down to at most one in each variable.

11.2.2 An Interactive Protcol for PSPACE

Using the arithmetization discussed earlier, we now show that the prover can convince the verifier in polynomial time.

First, the Prover claims to the Verifier that $P_{\Psi_0}() = 1$. At stage j of the interactive proof there will be some fixed values $r_1, \ldots, r_k \in \mathbb{F}$ chosen by the Verifier so far and a value $a_j \in \mathbb{F}$ for which the Prover will be trying to convince the Verifier that $P_{\Psi_j}(r_1, \ldots, r_k) = a_j$.

There are several different cases, depending on the form of Ψ_j.

$\Psi_j = \forall x_{k+1} \Psi_{j+1}$: In this case, the Prover computes $f_{j+1}(z) = P_{\Psi_{j+1}}(r_1, \ldots, r_k, z)$ and transmits the coefficients of f_{j+1}. The Verifier checks that $f_{j+1}(0) \cdot f_{j+1}(1) = a_j$ (which should be $P_{\Psi_j}(r_1, \ldots, r_k)$). If not, the Verifier rejects; otherwise, the Verifier chooses $r_{k+1} \in_R \mathbb{F}$ and sends r_{k+1} to the Prover. The new value $a_{j+1} = f_{j+1}(r_{k+1})$ and the protocol continues as the Prover tries to convince the Verifier that $P_{\Psi_{j+1}}(r_1, \ldots, r_{k+1}) = a_{j+1}$.

$\Psi_j = \forall x_{k+1} \Psi_{j+1}$: In this case, the Prover computes $f_{j+1}(z) = P_{\Psi_{j+1}}(r_1, \ldots, r_k, z)$ and transmits the coefficients of f_{j+1}. The Verifier checks that $f_{j+1}(0) \circledast f_{j+1}(1) = a_j$ (which should be $P_{\Psi_j}(r_1, \ldots, r_k)$). If not, the Verifier rejects; otherwise, the Verifier chooses $r_{k+1} \in_R \mathbb{F}$ and sends r_{k+1} to the Prover. The new value $a_{j+1} = f_{j+1}(r_{k+1})$ and the protocol continues as the Prover tries to convince the Verifier that $P_{\Psi_{j+1}}(r_1, \ldots, r_{k+1}) = a_{j+1}$.

$\Psi_j = R x_i \Psi_{j+1}$: In this case, the Prover computes $f_{j+1}(z) = P_{\Psi_{j+1}}(r_1, \ldots, r_{i-1}, z, r_{i+1}, \ldots, r_{k-1})$ and transmits the coefficients of f_{j+1}. (Unlike the other two cases there may be many coefficients and not just two coefficients.) The verifier checks that $(1 - r_i) f_{j+1}(0) + r_i f_{j+1}(1) = a_j$ (which should be $P_{\Psi_j}(r_1, \ldots, r_k)$). If not, the Verifier rejects; otherwise, the Verifier chooses $r'_i \in_R \mathbb{F}$ and sends r'_i to the Prover. The new value $a_{j+1} = f_{j+1}(r'_i)$ and the protocol continues as the Prover tries to convince the Verifier that $P_{\Psi_{j+1}}(r_1, \ldots, r_{i-1}, r'_i, r_{i+1}, \ldots, r_k) = a_{j+1}$.

In the base case when there are no quantifiers, the Verifier simply evaluates $P_\psi(r_1, \ldots, r_n)$ and accepts if and only if the result is correct.

The total number of stages is $n + n(n-1)/2$: one stage for each existential or universal quantifier, plus $\sum_{i=1}^{n} i$ stages for the R quantifiers. The maximum degree in any stage is no more than the greater of 2 and m, the number of clauses in ψ.

Clearly, if the values are correct, then the Prover can convince the Verifier at each stage by sending the correct polynomial for f_{j+1}.

In case $P_{\Psi_0}() \neq 1$, if the Verifier accepts then there is some stage at which $P_{\Psi_j}(\vec{r}) \neq a_j$ but $P_{\Psi_{j+1}}(\vec{r}') = a_{j+1}$. If the Prover sends the correct coefficients of f_{j+1} then the Verifier will immediately reject because the Verifier directly checks that the Prover's answers reflect the recursive definition of P_{Ψ_j}. If the Prover

sends the incorrect coefficients for f_{j+1} then the chance that the Verifier chooses a random value r on which $a_{j+1} = f_{j+1}(r) = P_{Psi_{j+1}}(\ldots, r, \ldots)$ is at most the degree of f_{j+1} divided by $|\mathbb{F}|$ which is at most $m/|\mathbb{F}|$.

By the union bound, the total failure probability is therefore less than:

$$\frac{(\frac{n(n-1)}{2} + n)m}{|\mathbb{F}|}$$

which for $|\mathbb{F}| \geq mn^3$ yields failure probability less than $1/n$. □

Lecture 12

Probablistically Checkable Proofs

May 13, 2004
Lecturer: Paul Beame
Notes: Chris Re

12.1 Probablisitically Checkable Proofs Overview

We know that $\mathsf{IP} = \mathsf{PSPACE}$. This means there is an interactive protocol between a prover P and a verifier V such that

$$\begin{aligned} \Psi \in \text{TQBF} &\rightarrow (P,V) \text{ accepts } \Psi \text{ with probability } 1 \\ \Psi \notin \text{TQBF} &\rightarrow (P,V) \text{ accepts } \Psi \text{ with Probability } \leq \frac{1}{2}. \end{aligned}$$

Suppose that we wanted to express the entire strategy of the prover in this protocol for all possible interactions with a verifier. This strategy would consist of a rooted tree of the possible interactions, where each node of the tree corresponds to the state of the system between rounds of communication and the edges are labelled by the values sent during the rounds. At each round, the verifier sends a polynomial number of random bits (in the size N of the input Ψ) so each node just before a verifier move has fan-out $2^{N^{O(1)}}$. Each node just before a prover move has fan-out 1 which lists the prover's (best) response to the previous communcation. The entire tree has depth $N^{O(1)}$. Because of the depth and fan-out of the tree, it has $2^{N^{O(1)}}$ nodes overall.

Thus, the IP protocol corresponds to a table of size $2^{N^{O(1)}}$ of which the protocol accesses/queries only $N^{O(1)}$ bits determined by the choice of $N^{O(1)}$ random bits. PCP is a generalization of these proof techniques along the two mentioned axes: the number of random bits and the number of queries allowed.

12.2 PCP definition

Think of a proof as a large table and the verification has access to only a small number of places in the table. This is the main idea of probabilistically checkable proofs.

Definition 12.1. $L \in \mathsf{PCP}(r(n), q(n)) \Leftrightarrow \exists$ a polynomial time oracle TM $V^?$ with access to $O(r(n))$ random bits and that makes $O(q(n))$ queries to its oracle such that

$$\begin{aligned} x \in L &\rightarrow \exists\, \Pi \text{ such that } \Pr_r[V^{\Pi}(x) \text{ accepts}] = 1 \\ x \notin L &\rightarrow \forall\, \Pi.\ \Pr_r[V^{\Pi}(x) \text{ accepts}] \leq \frac{1}{2} \end{aligned}$$

where Π denotes an oracle which we think of as a proof for L. Π is viewed as a table listing all the values on which it can be queried.

Upper bound on $|\Pi|$ How big does the proof table need to be? Think of a large table indexed by the queries and the random strings, this case implies it is at most $O(q(n))2^{O(r(n))}$. It follows that

Lemma 12.1. $\mathsf{PCP}(r(n), q(n)) \subseteq \mathsf{NTIME}(2^{O(r(n))}q(n))$.

A more general formulation There are still two constants in our formulation 1 and $\frac{1}{2}$. We can generalize PCP further by introducing the notions of the *soundness* and *completeness* of the proof system.

Definition 12.2. $L \in PCP_{c,s}(r(n), q(n)) \Leftrightarrow \exists$ a polynomial time oracle TM $V^?$ with access to $O(r(n))$ random bits and that makes $O(q(n))$ queries to its oracle such that

$$x \in L \rightarrow \exists\, \Pi \text{ such that } \Pr_r[V^{\Pi}(x) \text{ accepts}] \geq c \qquad \text{(Completeness)}$$

$$x \notin L \rightarrow \forall\, \Pi.\ \Pr_r[V^{\Pi}(x) \text{ accepts}] \leq s \qquad \text{(Soundness)}$$

This formulation is important for some hardness of approximation results. If the parameters are not specified we assume the original formulation.

Definition 12.3. $\mathsf{PCP}(\mathsf{poly}, \mathsf{poly}) = \bigcup_k \mathsf{PCP}(n^k, n^k)$

Remark. The definition of $\mathsf{PCP}(\mathsf{poly}, \mathsf{poly})$ was originally motivated by a different way of extending the idea of IP. This idea was to allow multiple all-powerful instead of just one prover. With such Multiprover Interactive Proof systems the multiple provers can be used with the restriction that they may not collude with each other. The set of languages proved in polynomial time in such settings was/is known as MIP. Later it was shown that the languages in MIP are precisely those in $\mathsf{PCP}(\mathsf{poly}, \mathsf{poly})$ and the oracle characterization was used but results are still sometimes stated as results about MIP.

12.2.1 Results about PCP

Immediately from the motivating discussion for the definition of the PCP classes above we have $\mathsf{IP} \subseteq \mathsf{PCP}(\mathsf{poly}, \mathsf{poly})$ and thus the following lemma.

Lemma 12.2. $\mathsf{PSPACE} \subseteq \mathsf{PCP}(\mathsf{poly}, \mathsf{poly})$.

As a corollary to Lemma 12.1 we have

Lemma 12.3. $\mathsf{PCP}(\mathsf{poly}, \mathsf{poly}) \subseteq \mathsf{NEXP}$ *and for any polynomial* $q(n)$ *we have* $\mathsf{PCP}(\log n, q(n)) \subseteq \mathsf{NP}$.

The main results about PCP are the following which show strong converses of the above simple inclusions.

Theorem 12.4 (Babai-Fortnow-Lund). $\mathsf{PCP}(\mathsf{poly}, \mathsf{poly}) = \mathsf{NEXP}$.

The following result is so strong it is known as *the* PCP Theorem. It was the culimination of a sequence improvements of the above result by Babai, Levin, Fortnow, and Szegedy, by Feige, Goldwasser, Lovasz, Safra, and Szegedy and by Arora and Safra.

Theorem 12.5 (Arora-Lund-Motwani-Szegedy-Sudan). $\mathsf{NP} = \mathsf{PCP}(\log \mathsf{n}, 1)$.

It is interesting to note that the $O(1)$ in the number of queries and can actually be reduced to 3 in the strongest versions of the PCP Theorem. This is important for many of the results about approximation problems. For example in approximating CLIQUE this stronger version is used to show that it is hard to approximate CLIQUE with even an $n^{1-o(1)}$ factor.

Note that the PCP Theorem can be seens as a strict strengthening of the result of Babai, Fortnow, and Lund since it yields the following corollary.

Corollary 12.6. $\mathsf{PCP}(\mathsf{poly}, 1) = \mathsf{NEXP}$.

Over the next several lectures we will go over the proofs of these theorems. To begin with we need a convenient characterization for NEXP.

12.2.2 A Complete Problem for NEXP

3SAT worked nicely as a complete problem for NP, so it is natural to look for an analog of 3SAT for NEXP. In fact, the Cook-Levin tableau argument will again be our basis for showing completeness.

The analogous problem will be an implicitly defined version of 3SAT. First we will define a problem ORACLE-3SAT that we will directly show to be NEXP-complete. ORACLE-3SAT will be 3SAT defined on exponential size formulas in 3CNF defined on 2^n variables and 2^m clauses.

Definition 12.4. An *oracle truth assignment* is a function $A : \{0,1\}^n \to \{0,1\}$ where we interpret $A(v) = 1 \Leftrightarrow x_v = 1$.

Definition 12.5. An *oracle 3CNF* is a map $C : \{0,1\}^m \to \{0,1\}^{3n+3}$ that specifies, given the index $w \in \{0,1\}^m$ of a clause, the 3 variables in that clause and the three signs for those variables. For convenience, the output $C(w)$ be represented by a tuple $(v_1, v_2, v_3, s_1, s_2, s_3)$ where $v_1, v_2, v_3 \in \{0,1\}^n$, $s_1, s_2, s_3 \in \{0,1\}$ and the clause represented is $x_{v_1}^{s_1} \vee x_{v_2}^{s_2} \vee x_{v_3}^{s_3}$ where x^0 denotes x and x^1 denotes $\neg x$.

Definition 12.6. An oracle 3CNF C is *satisfiable* if and only if there exists an oracle truth assignment A such that for all $w \in \{0,1\}^m$ there exists an $i \in \{1,2,3\}$ such that if $C(w) = (v_1, v_2, v_3, s_1, s_2, s_3)$ then $A(v_i) = s_i$.

Definition 12.7. We will represent oracle 3CNF formulas using multi-output combinational circuits C computing functions $C : \{0,1\}^m \to \{0,1\}^{3n+3}$. Therefore we define

$$\text{ORACLE-3SAT} = \{\langle C \rangle \mid C \text{ is a Boolean circuit representing a satisfiable oracle 3CNF}\}.$$

Lemma 12.7. ORACLE-3SAT *is* NEXP*-Complete*

Proof. Given a circuit C let m be the number of inputs to C and $3n + 3$ be the number of outputs of C. Given C, a NEXP machine can clearly guess and write down a truth assignment S for all 2^n choices of $v \in \{0,1\}^n$ and then verify that for all 2^m values of w, clause $C(w)$ is satisfied by A. Therefore ORACLE-3SAT $\in$ NEXP.

Let L be an arbitrary language in NEXP. Now consider the Cook-Levin tableau for L and how this converts first to a CIRCUIT-SAT problem and then to a 3SAT problem. For some polynomial p, The tableau has width $2^{p(|x|)}$ and height $2^{p(|x|)}$. The first $|x|$ entries in the first row depend on x; the remainder of

the first row are based on generic nondeterministic guesses y and in the entire rest of the tableau the tableau is very generic with each entry based on the 3 entries immediately above it in the tableau. Except for the first entries in this table the complexity of each local window depends only on the complexity of the Turing machine for L which is constant size and the only difference is the dependence of the first $|x|$ entries on x.

Now consider the 3CNF formula that results from the reduction of the CIRCUIT-SAT formula which simulates this tableau. Given the indices (i, j) in binary of a cell in the tableau it is easy to describe in polynomial time what the connections of the pieces of the circuit are that simulates this tableau and therefore what the pieces of the 3CNF formula are that will be produced as the 3CNF formula in the Cook-Levin proof. Thus we have a polynomial-time algorithm C' that, based on the index of a clause, will produce a 3-clause that is in an (exponential-size) 3CNF formula φ such that $x \in L$ if and only if φ is satisfiable. Since C' is a polynomial-time algorithm there is a polynomial size circuit C that simulates C'; moreover it is very easy to produce C given the input x and the description of the Turing machine for L. The reduction maps x to $\langle C \rangle$. Clearly $x \in L$ if and only if $\langle C \rangle \in$ ORACLE-3SAT. □

ORACLE-3SAT is slightly inconvenient to use for our purposes so we will use a variant of the idea that includes as a single Boolean function both the output of the circuit and the verification of the 3-CNF clause that it outputs. Suppose that C is a boolean circuit with g gates. Consider a function B which takes as input: the original input w to C, a triple of 3 variables v_1, v_2, v_3 output by C and purported values of each of the gates of C and 3 binary values a_1, a_2, a_3 and states that *if* on input w, the variables appearing in the output clause of C are correctly represented in the input description for B and the values of the gates of C are correctly represented in the input description to B *then* the signs of the clause output by C on input w will evaluate to true if variable $x_{v_1} = a_1$, $x_{v_2} = a_2$ and $x_{v_3} = a_3$. Because we have included the values of the internal gates of C we can easily express B as a Boolean formula based on C.

More formally: $B : \{0,1\}^{m+g+3n+3} \to \{0,1\}$ is defined as follows where $|w| = m$, $|z| = g$, $|v_i| = n$, and $|a_i| = 1$.

$$B(w, z, v_1, v_2, v_3, a_1, a_2, a_3)$$
$$= \bigvee_{s_1,s_2,s_3} (C(w) = (v_1, v_2, v_3, s_1, s_2, s_3) \text{ and } z \text{ represents values of the gates of } C) \to \exists i.\ (s_i = a_i)$$

The fact that B can represented as a Boolean formula simply mirrors the usual argument converting CIRCUIT-SAT to SAT.

Notice that the definition of B implies that for an oracle assignment A,

$$A \text{ satisfies } C \Leftrightarrow \forall w, z, v_1, v_2, v_3\ B(w, z, v_1, v_2, v_3, A(v_1), A(v_2), A(v_3)).$$

Definition 12.8. A Boolean formula B in $h + 3n + 3$ variables is a *satisfiable implicit 3CNF formula* iff there is an oracle truth assignment $A : \{0,1\}^n \to \{0,1\}$ such that $\forall w' \in \{0,1\}^h$, $\forall v_1, v_2, v_3 \in \{0,1\}^n$, $B(w', v_1, v_2, v_3, A(v_1), A(v_2), A(v_3))$ is true.

Definition 12.9. IMPLICIT-3SAT $= \{\langle B \rangle \mid B \text{ is a satisfiable implicit 3CNF formula } \}$.

Clearly the conversion from $\langle C \rangle$ to $\langle B \rangle$ as defined above is polynomial time so we have that ORACLE-3SAT is polynomial time reducible to IMPLICIT-3SAT and thus:

Theorem 12.8. IMPLICIT-3SAT *is* NEXP*-Complete.*

Now that we have a complete problem for NEXP, we need to show how to convince ourselves in the required time bounds to achieve our goal of showing that NEXP = PCP(poly, poly).

12.2.3 Verification procedure for an oracle truth assignment

1st idea The definition of the satisfiability of the implicit 3CNF B can easily be seen to be computation that can be done in coNP^A since there are only universal quantifiers other than the oracle calls and the evaluation of B. Since $\mathsf{coNP}^A \subseteq PSPACE^A$ we could try to modify the IP=PSPACE to use an oracle.

This IP procotol relied on our ability to arithmetize formulas, like quantified versions of B that give values over $\{0,1\}$, to polynomials over a field $\mathbb{F}$ of moderate size. It then relied on the ability to query such functions on randomly chosen field elements $r \in \mathbb{F}$. In trying to apply the protocol here there is no problem with arithmetizing B. However, so far, we only have an oracle A that gives Boolean outputs given an input string $v \in \{0,1\}^n$; we would need to be able to evaluate A on elements of $\mathbb{F}^n$ instead.

Fortunately, we can do this by using a *multilinear* extension of A.

Definition 12.10. A function in n variables is multilinear if and only if it can be expressed as a multivariate polynomial which has degree at most 1 in each variable.

Lemma 12.9. *For any $A : \{0,1\}^n \to \mathbb{F}$ there is a (unique) multilinear polynomial that extends A. That is there exists a multilinear $\widehat{A} : \mathbb{F}^n \to \mathbb{F}$ such that $\widehat{A}(v) = A(v)$ for $v \in \{0,1\}^n$.*

Proof. Let $\widehat{A}(x) = \sum_{v\in\{0,1\}^n} A(v) \prod x_i \prod(1-x_i)$. This is the desired multilinear polynomial. Uniqueness is left as an exercise. □

So we have two parts for the proof table so far: A table of the multilinear oracle $\widehat{A}$ and the full interaction tree for the $\mathsf{IP}^{\widehat{A}}$ protocol for the $\mathsf{coNP}^{\widehat{A}}$ problem of verifying that $\widehat{A}$ satisfies B.

However, this is not enough. The prover might not produce a correct table of $\widehat{A}$! Thus, in addition to $\widehat{A}$, the proof table must include part that allows the verifier to check with some certainty that $\widehat{A}$ really is a multilinear extension of a truth assignment.

Actually, because the prover can only check a small part of the table there will be no way to convince the verifier that the table $\widehat{A}$ really is multilinear. However, as we will see in the next lecture, we just need it to be close to such an extension which is something the verifier will be able check.

Lecture 13

PCP Continued

May 13, 2004
Lecturer: Paul Beame
Notes: Tian Sang

Last time we began the proof of the theorem that $\mathsf{PCP}(\mathsf{poly}, \mathsf{poly}) = \mathsf{NEXP}$.

We showed that IMPLICIT-3SAT is NEXP-complete where IMPLICIT-3SAT takes as input a Boolean formula B defined on $m' + 3n + 3$ variables where $m' = m + g$ and B is in the language if and only if there is an oracle truth assignment $A : \{0,1\}^n \to \{0,1\}$ such that

$$\exists A \forall w, z, v_1, v_2, v_3 \; B(w, z, v_1, v_2, v_3, A(v_1), A(v_2), A(v_3)).$$

(We interpreted w as an m-bit long clause index, z as gate variables, each v_i as an n-bit long variable index in the clause indexed by w, and $A(v_i)$ is the truth assignment to variable x_{v_i}.)

Given A, the verification that A satisfies B is in coNP^A. This could be viewed as an oracle special case of the $\mathsf{IP} = \mathsf{PSPACE}$ protocol. An alternative to this is a sum-check protocol of Lund, Fortnow, Karloff, and Nisan protocol for $\#\mathsf{P}$ (given in Sipser's text and discussed in the next lecture) which instead verifies the value of

$$\sum_{w,z,v_1,v_2,v_3} B(w, z, v_1, v_2, v_3, A(v_1), A(v_2), A(v_3)).$$

In either case we need to be able to convert B to a low degree polynomial in w, z, v_1, v_2, v_3 over $\mathbb{F}$ and evaluate that polynomial on random variables in $\mathbb{F}$ instead of $\{0,1\}$. To do this we needed to be able to evaluate A on such assignments so we use the fact shown last time that there is a (unique) multilinear extension of an assignment $A : \{0,1\}^n \to \mathbb{F}$. Let $\overline{A} : \mathbb{F}^n \to \mathbb{F}$ be this multilinear extension.

Thus the proof table gives values for $\overline{A}$, as well as a full tree of all possible executions of the prover's strategy for a $\mathsf{coNP}^{\overline{A}}$ ($\mathsf{IP} = \mathsf{PSPACE}$ style) proof that A is a satisfying assignment to $\forall w, z, v_1, v_2, v_3 \; B(w, z, v_1, v_2, v_3, A(v_1), A(v_2), A(v_3))$. Such a tree is given in Figure 13.1.

This would be OK if the proof table correctly gives an $\overline{A}$ that really is multilinear. However, since the verifier only will examine a polynomial number of places out of the exponentially many in the proof the verifier can never be sure that the table truly is multilinear. The verifier will only be able to verify that the table is close (in Hamming distance) to a multilinear function.

A useful property of the $\mathsf{IP} = \mathsf{PSPACE}$ protocol (and the $\#\mathsf{P}$ protocol) is that the final polynomial is evaluated only once on random inputs chosen by the verifier. Thus on any run, the verifier will only examine $\overline{A}$ in 3 places, randomly chosen by the verifier.

If $\overline{A}$ instead is merely close to a multilinear $\widehat{A} : \mathbb{F}^n \to \mathbb{F}$, such that say $dist(\overline{A}, \widehat{A}) \leq \delta$, then with probability $\geq 1 - 3\delta$ the verifier will only see values on which $\overline{A}$ and $\widehat{A}$ agree and thus the additional error contributed to the acceptance probability of the protocol by the difference between $\overline{A}$ and $\widehat{A}$ is at most 3δ.

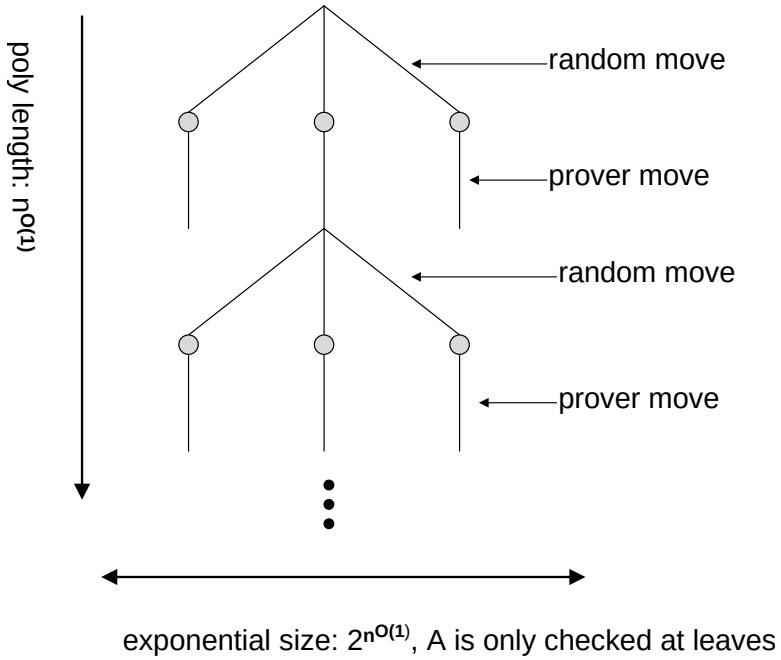

Figure 13.1: Proof Tree Corresponding to an Interactive Protocol

So we need to check that $dist(\overline{A}, \widehat{A}) \leq \delta$ for some multilinear function $\widehat{A}$. This is done using a multilinearity test that is a special case of a test that checks whether or not a polynomial has degree at most k in each variable.

13.1 The Max-degree-k test

Definition 13.1. $f : I^n \to \mathbb{F}$ is called *max-degree-k* iff it can be extended to $f : \mathbb{F}^n \to \mathbb{F}$, that has degree at most k in each variable.

Definition 13.2. For $u_1, \ldots, u_n \in I^n$ an *i-line* is a set $\{(u_1, \ldots, u_{i-1}, z, u_{i+1}, \ldots, u_n) \mid z \in I)\}$.

The following test is a generalization to a max-degree-k test of a multilinearity test due to Feige, Goldwasser, Lovasz, Safra, and Szegedy that improved on the original multilinearity test used by Babai, Fortnow, and Lund. The following analysis is from Friedl, Hatsagi, and Shen.

Aligned Line Test: Choose $k+1$ distinct elements $a_1, \ldots, a_{k+1} \in I$
Repeat t times:

(1) Choose $i \in_R \{1, \ldots, n\}$

(2) Choose a random point $(u_1, \ldots, u_n) \in_R |I|^n$

(3) Check that on the i-line through $(u_1, \ldots, u_n)$ at $u_i, a_1, \ldots, a_{k+1}$, f looks like a degree $\leq k$ polynomial, if not, reject. That is, check that f on the $k+2$ points $(u_1, \ldots, u_n), (u_1, \ldots, u_{i-1}, a_1, u_{i+1}, \ldots, u_n), \ldots, (u_1, \ldots, u_{i-1}, a_{k+1}, u_{i+1}, \ldots, u_n)$ fits a degree k polynomial evaluated at $u_i, a_1, \ldots, a_{k+1}$.

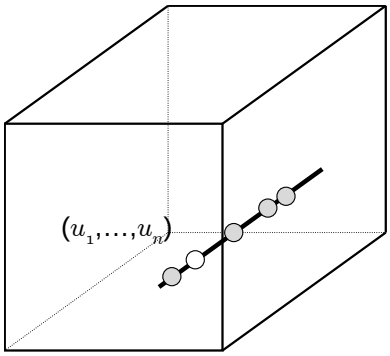

Figure 13.2: Aligned Line Test

In step (3) it would be more natural simply to check f on $k+2$ random points on a randomly chosen i-line but this version is both easier to analyze and uses less randomness.

Definition 13.3. Let $P(n,k) = \{f : I^n \to \mathbb{F}$ that are max-degree-k $\}$ and $P_i(n,k) = \{f : I^n \to \mathbb{F}$ that has degree $\leq k$ in $x_i\}$. Observe that $P(n,k) = \bigcap_{i=1}^{k} P_i(n,k)$.

Definition 13.4. Define $d(f, P(n,k)) = Pr_{x \in_R I^n}[f(x) \neq g(x)]$, and $d(f,S) = \min_{g \in S} d(f,g)$.

Lemma 13.1. *In a single round of the aligned line test,* $\Pr[$*test rejects* $\mid i$ *is chosen*$] \geq d(f, P_i(n,k))$.

Proof. For each choice of $u_1, \ldots, u_{i-1}, u_{i+1}, \ldots, u_n$, there is a unique degree k polynomial $h_{u_1,\ldots,u_{i-1},u_{i+1},\ldots,u_n}(z)$ that equals $f(u_1, \ldots, u_{i-1}, z, u_{i+1}, \ldots, u_n)$ for $z = a_1, \ldots, a_{k+1}$. The probability that the test does not reject in step (3) is the probability that for $u_i \in_R I$, $f(u_1, \ldots, u_{i-1}, u_i, u_{i+1}, \ldots, u_n) = h_{u_1,\ldots,u_{i-1},u_{i+1},\ldots,u_n}(u_i)$. Combining all these $h_{u_1,\ldots,u_{i-1},u_{i+1},\ldots,u_n}$ functions for different values of $u_1, \ldots, u_{i-1}, u_{i+1}, \ldots, u_n$ this yields a function $h \in P_i(n,k)$ and the probability that the test does not reject is the probability that on a random $\overrightarrow{u} = (u_1, \ldots, u_n) \in I^n$, $f(\overrightarrow{u}) = h(\overrightarrow{u})$ and this is precisely $1 - d(f,h) \leq 1 - d(f, P_i(n,k))$. □

The correctness of the test is based on the lemma above and the following analysis.

Lemma 13.2. *For any function* $f : I^n \to \mathbb{F}$ *and any* k*,*

$$d(f, P(n,k)) \leq 6(k+1) \sum_{i=1}^{n} d(f, P_i(n,k)) + 2nk/\sqrt{|I|}.$$

We first see how this implies that the aligned line test successfully detects functions that are far from max-degree k.

Corollary 13.3. *If* $d(f, P(n,k)) \geq \delta \geq 4nk/\sqrt{|I|}$ *then running the aligned test for* $t = \Theta(nk/\delta)$ *rounds will ensure that the test rejects with arbitrarily high constant probability. The total number of queries is* $\Theta(nk^2/\delta)$ *and the total number of random bits is* $\Theta(\frac{n^2k}{\delta} \log |I|)$.

Proof. By Lemma 13.1,

$$\begin{aligned}\Pr[\text{test rejects in a round}] &\geq \frac{1}{n}\sum_{i=1}^{n}\frac{d(f,P_i(n,k))}{n}\\ &\geq \frac{d(f,P(n,k))-\frac{2nk}{\sqrt{|I|}}}{6(k+1)n}\\ &\geq \frac{\delta-\frac{2nk}{\sqrt{|I|}}}{6(k+1)n}\\ &\geq \frac{\delta}{12(k+1)n},\end{aligned}$$

by the assumption on δ. Thus the expected number of rounds before the test rejects is at most $12(k+1)n/\delta$. Repeating this $t = \Omega(kn/\delta)$ rounds yields arbitrarily high constant probability of detection. The aligned line test makes $k+1$ queries per round and uses $n\log_2|I|$ random bits per round. □

Proof of Theorem 12.4. This Corollary is enough to complete the proof that $\mathsf{PCP}(\mathsf{poly},\mathsf{poly}) = \mathsf{NEXP}$. Using $\delta = 1/10$, say, and using a field $\mathbb{F}$ with $I \subseteq \mathbb{F}$ and $|I| \geq 160n^2$ (since $k = 1$ in the application), running the aligned line test for $O(n)$ rounds (which yields $O(n)$ queries and $O(n^2\log n)$ random bits is sufficient to ensure that with probability $\geq 9/10$, $\overline{A}$ is within Hamming distance δ of some multilinear $\widehat{A}$. Using the proof table for a $\mathsf{coNP} \subseteq \mathsf{IP}$ (oracle) protocol with error at most $1/10$ to verify that $\overline{A}$ satisfies B, yields total failure probability at most $1/10 + 1/10 + 3\delta = 1/2$. □

We now sketch some of the ideas involved in the proof of Lemma 13.2. The main idea behind all the low degree tests we will use is the following generalization to multivariate polynomials of the fact that a low degree polynomial only has a small number of roots.

Lemma 13.4 (Schwartz, Zippel). *If $p \in \mathbb{F}[x_1,\ldots,x_n]$ is a polynomial of total degree $\leq d$, and $p \neq 0$, then for $a_1,\ldots,a_n \in_R I \subseteq \mathbb{F}$, $\Pr[p(a_1,\ldots,a_n)=0] \leq \frac{d}{|I|}$*

Proof. By induction on n. The base case $n = 1$ follows because the polynomial p has $\leq d$ roots.

For the induction step, write

$$p(x_1,\ldots,x_n) = \sum_{i=0}^{m} p_i(x_1,\ldots,x_{n-1})x_n^i.$$

Since the total degree of p is at most d, the total degree of p_i is at most $d-i$.

$$\begin{aligned}\Pr[p(a_1,\ldots,a_n)=0] &\leq \Pr[p_m(a_1,\ldots,a_{n-1})=0] + \Pr[p(a_1,\ldots,a_n)=0 \mid p_m(a_1,\ldots,a_{n-1})\neq 0]\\ &\leq \frac{d-m}{|I|}+\frac{m}{|I|} = \frac{d}{|I|}\end{aligned}$$

where the bound for the first term follows from the inductive hypothesis applied to p_m and the bound for the second term follows from the application of the base case to the polynomial $q(x_n) = \sum_{i=0}^{m} p_i(a_1,\ldots,a_{n-1})x_n^i$. □

Corollary 13.5. *If $|I| > 3nk$ and $d(f,P(n,k)) \leq 1/3$ then there is a unique g such that $d(f,g) = d(f,P(n,k))$*

Proof. Let $g \in P(n,k)$ be a polynomial witnessing the fact that $d(f, P(n,k)) < 1/3$ so $d(f,g) < 1/3$. Suppose $h \in P(n,k)$ and $h \neq g$. The total degree of each of g and h is at most nk. Applying the Schwartz-Zippel Lemma to $g - h$ we see that $d(g,h) \geq 1 - \frac{nk}{|I|} > \frac{2}{3}$. Since by the triangle inequality $d(g,h) \leq d(f,g) + d(f,h) \leq 1/3 + d(f,h)$, we obtain that $d(f,h) > d(g,h) - 1/3 > 2/3 - 1/3 = 1/3$ implying that g is unique. □

Let $f^i \in P_i(n,k)$ be such that $d(f, f^i) = d(f, P_i(n,k))$. Observe that by definition, $d(f^i, g) = \frac{1}{|I|}\sum_{c \in I} d(f^i|_{x_i=c}, g|_{x_i=c})$ and that if $g \in P(n,k)$ then $g|_{x_i=c} \in P(n-1,k)$ for any c. In particular this means that $d(f^i, g) \geq \frac{1}{|I|}\sum_{c \in I} d(f^i|_{x_i=c}, P(n-1,k))$.

The following is an immediate consequence of Corollary 13.5.

Lemma 13.6. *If $|I| > 3nk$ and $f^i \in P_i(n,k)$ and $g \in P(n,k)$ agree on at least 2/3 of all i-lines then $d(f^i, g) = d(f, P(n,k))$ and for all $c \in I$, $d(f^i|_{x_i=c}, g|_{x_i=c}) = d(f^i|_{x_i=c}, P(n-1,k))$ and thus*

$$d(f^i, g) = \frac{1}{|I|}\sum_{c \in I} d(f^i|_{x_i=c}, P(n-1,k)).$$

For any $k+1$ hyperplanes $x_i = c_1, \ldots, x_i = c_{k+1}$, the fraction of all i-lines on which f^i and g disagree is at most $\sum_{j=1}^{k+1} d(f^i_{x_i=c_j}, g)$ since f and g are in complete agreement on any i-line on which f^i and g agree on all of these hyperplanes. (This follows because f^i and g are both degree k polynomials along any i-line; see Figure 13.3.) For the same reason, on any i-line on which f^i and g disagree, they agree on at most k points.

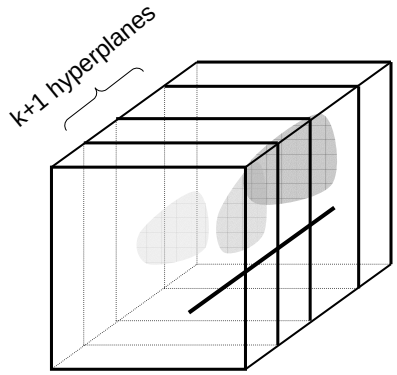

Figure 13.3: Behavior on i-lines related to $k+1$-hyperplanes

Moreover, even if there are $k+1$ hyperplanes $x_i = c_1, \ldots, x_i = c_{k+1}$ on which $\sum_{j=1}^{k+1} d(f^i_{x_i=c_j}, P(n-1,k)) \leq 1/3$ then letting $g_1, \ldots, g_{k+1} \in P(n-1,k)$ be the witnessing polynomials close to $f^i_{x_i=c_j}$ then we can interpolate the g_j to a single polynomial $g' \in P(n,k)$ for which f^i and g' agree on a fraction $1 - \alpha \geq 2/3$ of all i-lines, $d(f^i, g') = d(f^i, P(n,k)) \leq \alpha$, and $d(f^i_{x_i=c}, P(n-1,k)) = d(f^i_{x_i=c}, g'_{x_i=c})$ for all $c \in I$. Again, on the α fraction of i-lines on which f^i and g' do not completely agree, there are at most k points of agreement, so in total there are at most an $\alpha k/|I| \leq k/(3|I|)$ fraction of all points in I^n that could contribute to places where $d(f^i_{x_i=c}, P(n-1,k)) < \alpha$. By an averaging argument we obtain the following lemma.

Lemma 13.7. *For $|I| > 3nk$ for any $\mu > 0$, either*
(a) $\Pr_{c \in_R I}[d(f^i|_{x_i=c}, P(n-1,k)) \leq 1/(3k+3)] \leq \frac{k+1}{|I|}$ *or*
(b) $\Pr_{c \in_R I}[d(f^i|_{x_i=c}, P(n-1,k)) \leq d(f^i, P(n,k)) - \mu] \leq \frac{k}{3\mu|I|}$.

The final argument follows, for suitable μ, using induction by iteratively choosing $c_i \in_R I$ and setting each $x_i = c_i$ for $i = 1, \ldots, n$. We begin by observing that $d(f, P(n,k)) \leq d(f, f^1) + d(f^1, P(n,k))$ and expanding $d(f^1, P(n,k))$. Intuitively, at each step, either (a) holds and except for a $\frac{k+1}{|I|}$ fraction of choices so far, the average distance between f^i and a max-degree k polynomial on the remaining choices is at least $1/(3k+3) \geq d(f, P(n,k))/(3k+3)$, or (b) holds and except for a $\frac{k}{3\mu|I|}$ fraction of choices so far, the distance between f^i and a max-degree k polynomial is well represented by its error on the remaining choices (except for an additive error of at most μ). Furthermore, the distance at each choice c_i is at most the sum of the error based on setting one more variable in f^i, $d(f^i, f^{i+1})$, and the distance between f^{i+1} and a max-degree k polynomial in the remaining variables. Since after all values are set, the function defined on a single input can be exactly represented by a max-degree k polynomial if (b) always holds then

$$d(f^1, P(n,k)) \leq n\mu + \sum_{i=1}^{n-1} d(f^i, f^{i+1}) + n\frac{k}{3\mu|I|} \leq n\mu + \sum_{i=1}^{n-1}(d(f, f^i) + d(f, f^{i+1})) + n\frac{k}{3\mu|I|}$$

and thus

$$\begin{aligned} d(f, P(n,k)) &\leq 2\sum_{i=1}^{n} d(f, f^i) + n\mu + \frac{nk}{3\mu|I|} \\ &= 2\sum_{i=1}^{n} d(f, P_i(n,k)) + n\mu + \frac{nk}{3\mu|I|}. \end{aligned}$$

However, if (a) holds at some stage, consider the last stage j in which (a) holds. Except for a $(k+1)/|I|$ fraction of inputs, $d(f, P(n,k))/(3k+3)$ is a lower bound on the distance between f^j and a max-degree k polynomial on the remaining inputs, and thus

$$\begin{aligned} d(f, P(n,k))/(3k+3) &\leq (n-j)\mu + \sum_{i=j}^{n-1}(d(f, f^i) + d(f, f^{i+1})) + (n-j)\frac{k}{3\mu|I|} + \frac{k+1}{|I|} \\ &= 2\sum_{i=j}^{n} d(f, P_i(n,k)) + (n-j)\mu + \frac{(n-j)k}{3\mu|I|} + \frac{k+1}{|I|}. \end{aligned}$$

Strictly speaking, the stages at which (a) holds depend on the choices of the c_i so this argument is not fully rigorous as stated; however, it can be made rigorous by maintaining the sequence of choices made explicitly and summing appropriately. The bounds in Lemma 13.2 follow.

Lecture 14

PCP and NP

May 20, 2004
Lecturer: Paul Beame
Notes: Trent Moore

Last time we finished the proof of Babai, Fortnow, and Lund's theorem that $\mathsf{PCP}(\mathsf{poly}, \mathsf{poly}) = \mathsf{NEXP}$. The following is an almost immediate corollary based on scaling down the parameters in the proof to NP instead of NEXP.

Corollary 14.1. $\mathsf{NP} \subseteq \mathsf{PCP}(\mathsf{polylog}, \mathsf{polylog})$.

We will prove the following somewhat stronger theorem of Babai, Fortnow, Levin, and Szegedy that yields polynomial-size proofs and a theorem by Feige, Goldwasser, Lovasz, Safra, and Szegedy that does not yield polynomial-size proofs but has much better parameters.

Theorem 14.2 (Babai-Fortnow-Levin-Szegedy). $\mathsf{NP} \subseteq \mathsf{PCP}(\mathsf{polylog}, \mathsf{polylog})$*; moreover the proof table for the* $\mathsf{PCP}(\mathsf{polylog}, \mathsf{polylog})$ *algorithm is of polynomial size.*

Theorem 14.3 (Feige-Goldwasser-Lovasz-Safra-Szegedy). $\mathsf{NP} \subseteq \mathsf{PCP}(\log n \log\log n, \log n \log\log n)$.

We will go through one proof that will yield both theorems. Before doing so, we will prove a connection between PCPs for NP and MAX-CLIQUE.

Definition 14.1. For an undirected graph G, MAX-CLIQUE(G) (also known as $\omega(G)$) is the size of the largest clique in G. An function f is a factor α approximation algorithm for MAX-CLIQUE iff MAX-CLIQUE$(G)/\alpha \leq f(G) \leq$ MAX-CLIQUE(G).

Theorem 14.4 (Feige-Goldwasser-Lovasz-Safra-Szegedy). *If* $\mathsf{NP} \subseteq \mathsf{PCP}(r(n), q(n))$ *and there is a polynomial time algorithm approximating* MAX-CLIQUE *better than a factor of* 2 *then* $\mathsf{NP} \subseteq \mathsf{DTIME}(2^{O(r(n)+q(n))})$.

Proof. Let $L \in \mathsf{NP}$ and $V_L^?$ be a polytime verifier in a $\mathsf{PCP}(r(n), q(n))$ protocal for L.

A *transcript of* $V_L^?$ can be described by a random string $r \in \{0,1\}^{r'(n)}$ where $r'(n)$ is $O(r(n))$ and the pairs (q_i, a_i) of queries and answers from the oracle for $i = 1, \ldots, q'(n)$, where $q'(n)$ is $O(q(n))$, and $a_i \in \{0,1\}$.

On input x, transcript t is *accepting* if and only if $V_L^?(x, r)$ given oracle answers $(q_1, a_1), \ldots, (q_{|x|}, a_{|x|})$ accepts. Two transcripts $t = (r, q, a)$, $t' = (r', q', a')$ are *consistent* if and only if $\forall i, j$ if $q_i = q_j$ then $a_i = a'_j$.

There are $2^{r'(n)+q'(n)}$ total transcripts on an input x with $|x| = n$ since the queries q_i are determined by the random string and the previous a_i's. Define a graph G_x with

$$V(G_x) = \{ \text{ accepting transcripts of } V_L^? \text{ on input } x\}.$$

In G_x, $t, t' \in V(G_x)$ are connected by an edge if and only if t, t' are consistent accepting transcripts. Since $V_L^?$ is polynomial time, we can verify whether a transcript is in $V(G_x)$ and check any edge of $E(G_x)$ in polynomial time. (Note: There is no need to write out the q_i part in creating the graph since the q_i can be determined as above.)

Observe that a clique in G_x corresponds precisely to all accepting computations based on a single oracle. In one direction, if an oracle is fixed then all accepting computations given that oracle will have consistent transcripts. In the other direction, for a clique in G_x, any oracle query yields the same answer on all the transcripts in the clique and therefore we can extend those answers consistently to a single oracle for which the transcripts in the clique correspond to accepting computations on that oracle.

Therefore

$$\max_{\Pi} \Pr[V_L^{\Pi}(x, r) \text{ accepts }] = \text{MAX-CLIQUE}(G_x)/2^{r'(n)}.$$

(As a sanity check, notice that there are at most $2^{r'(n)}$ mutually consistent transcripts; otherwise there would be two consistent transcripts with the same random strings and different query answers and these must be inconsistent with each other.)

Therefore by the PCP definition, if $x \in L$ then G_x has a clique of size $2^{r'(n)}$ but if $x \notin L$ then G_x has a clique of size at most $2^{r'(n)}/2$. The algorithm for L simply runs the approximation algorithm on input G_x and accepts if the answer is larger than $2^{r'(n)}/2$. The running time is polynomial is the size of G_x which is $2^{O(r(n)+q(n))}$. □

Proof of Theorems 14.2 and 14.3. The proof will follow similar lines to that of Babai, Fortnow, and Lund. Given a 3-CNF formula φ we can express φ implicitly as a formula in fewer variables just like we did using the B formula. The following table summarizes the similarities and represents the scaled down parameters.

Before [BFL]	Now
2^n vars indexed by n bits	n vars indexed by $\log n$ bits
2^m clauses indexed by m bits	$\leq 8n^3$ 3-clauses indexed by $\ell = 3\log n + 3$ bits
$A : \{0,1\}^n \to \{0,1\}$	$a : \{0,1\}^{\log n} \to \{0,1\}$
$\lvert\mathbb{F}\rvert$ polynomial in n	$\lvert\mathbb{F}\rvert$ polylog in n

In order to understand things better we will now express the proof table explicitly and use the sum-check protocol for the verification because it is more explicit. Let $i_1, i_2, i_3 \in \{1, \ldots, n\}$ be indices of clauses. (We use i instead of v to emphasize their size.) We can view the 3-CNF formula φ as a map $\widehat{\varphi} : \{0,1\}^{\ell} \to \{0,1\}$ saying which of the $8n^3$ possible clauses appear in φ. That is,

$$\widehat{\varphi}(i_1, i_2, i_3, s_1, s_2, s_3) = 1 \iff \text{the clause denoted } (x_{i_1} = s_1) \vee (x_{i_2} = s_2) \vee (x_{i_3} = s_3) \text{ is in } \varphi.$$

φ is satisfiable iff there is an assignment $a : \{0,1\}^{\log n} \to \{0,1\}$ such that

$$\forall (i_1, i_2, i_3, s_1, s_2, s_3) \in \{0,1\}^{\ell} (\widehat{\varphi}(i_1, i_2, i_3, s_1, s_2, s_3) = 0 \text{ or } a(i_1) = s_1,\ a(i_2) = s_2,\ \text{or } a(i_3) = s_3).$$

Let $\bar{a}$ and $\bar{\varphi}$ be multilinear extensions of a and $\widehat{\varphi}$. Then

$$\varphi \text{ is satisfiable} \iff \forall (i_1, i_2, i_3, s_1, s_2, s_3) \in \{0,1\}^{\ell}\ \bar{\varphi}(i_1, i_2, i_3, s_1, s_2, s_3) \cdot (\bar{a}(i_1) - s_1) \cdot (\bar{a}(i_2) - s_2) \cdot (\bar{a}(i_3) - s_3) = 0.$$

In polynomial time the verifier can easily produce $\bar{\varphi}$ on its own. The verifier will run a multilinearity test on $\bar{a}$ as before. Let $y = (i_1, i_2, i_3, s_1, s_2, s_3)$ and for any function a let

$$CC(y, a) = \bar{\varphi}(y) \cdot (a(i_1) - s_1) \cdot (a(i_2) - s_2) \cdot (a(i_3) - s_3)$$

be the correctness check for clause y and assignment a. Thus φ is satisfiable if and only if $\forall y \in \{0,1\}^\ell$ $CC(y,\bar{a}) = 0$. Observe that the verifier can efficient produce the polynomial for CC and since $\bar{\varphi}$ is multilinear, if $\bar{a}$ is multilinear then $CC(y,\bar{a})$ has degree at most 2 in each variable and total degree at most 4.

We cannot probe all 2^ℓ possible choices y. Instead for each assignment $\bar{a}$ we can think of the possible values of $CC(y,\bar{a})$ as a vector of length 2^ℓ which we want to be the vector 0^{2^ℓ}. The idea is to use a linear error-correcting code (that maps 0^{2^ℓ} to a zero vector) and any non-zero vector to a vector that has a constant fraction of non-zero entries so we will be able to detect whether or not the original vector was 0^{2^ℓ} with reasonable probability. The code we use here is a Reed-Muller code but many other codes would also work.

Choose ℓ random elements $R = (r_1, r_2, \ldots, r_\ell) \in_R I^\ell$ where $I \subseteq \mathbb{F}$ and define

$$E_R(a) = \sum_{y \in \{0,1\}^\ell} CC(y,a) \prod_{i \text{ s.t. } y_i = 1} r_i.$$

For a fixed a and varying R, $E(a) = E(a, r_1, \ldots, r_\ell)$ is a multilinear polynomial in $r_1, r_2, \ldots, r_\ell$ with coefficients $CC(y,a)$. Moreover, $E(a)$ is the zero polynomial if and only if $\forall y \in \{0,1\}^\ell$ $CC(y,a) = 0$. By the Schwartz-Zippel Lemma applied to $E(a)$, $\Pr_R[E_R(a) = 0] \leq \frac{\ell}{|I|}$.

To check that φ is satisfiable, the verifier chooses a random R and checks that $E_R(\bar{a}) = 0$ using the following protocol.

Sum-Check interactive protocal of Lund-Fortnow-Karloff-Nisan: Given a multivariable polynomial p of max degree k verify $\sum_{y \in \{0,1\}^\ell} p(y) = c_0$ (where in our case, $c_0 = 0, k = 2$, and $p(y) = CC(y,\bar{a}) \prod_{i \text{ s.t. } y_i = 1} r_i$).

Define $g_1(z) = \sum_{y_2,\ldots,y_\ell \in \{0,1\}} p(z, y_2, \ldots, y_\ell)$.

The prover sends the coefficients of a degree k polynomial f_1 claimed to be g_1. The verifier checks that $f_1(0) + f_1(1) = c_0$, chooses random $r_1' \in_R I \subseteq \mathbb{F}$, sends r_1' to prover, and sets $c_1 = f_1(r_1')$.

At the next round, $g_2(z) = \sum_{y_3,\ldots,y_\ell \in \{0,1\}} p(r_1', z, y_2, \ldots, y_\ell)$, the prover sends the coefficients of f_2, the check is that $f_2(0) + f_2(1) = c_1$ and so forth.

At the end, the verifier directly checks that the value of $p(r_1', r_2', \ldots, r_\ell') = c_\ell$.

In the case of applying the sum-check protocol for checking that

$$E_R(\bar{a}) = \sum_{y \in \{0,1\}^\ell} CC(y,\bar{a}) \prod_{i \text{ s.t. } y_i = 1} r_i = 0,$$

the values of $r_1, \ldots, r_\ell$ are known to the verifier. Once the values of $y_1, \ldots, y_\ell$ have been substituted by $r_1', \ldots, r_\ell'$, the structure of $CC(y,a)$ ensures that $\bar{a}$ will only need to be queried in the three random places specified by $r_1', \ldots, r_{\ell-3}'$. Thus the final check of the verifier can be done by the verifier with three queries to the purported multilinear extension $\bar{a}$ of a.

Proof Table: The above protocol is described interactively. However, the proof yields the following entries in the proof table. For each $r_1', r_2', \ldots, r_{i-1}' \in I$, for $1 \leq i \leq \ell$ the table contains coefficients of a degree $\leq k$ polynomial, $g_{r_1',\ldots,r_{i-1}'}(z) = \sum_{y_{i+1},\ldots,y_\ell} p(r_1', \ldots, r_{i-1}', z, y_{i+1}, \ldots, y_\ell)$.

The size of the proof table is $O(|I|^\ell \cdot k \cdot \log |I|)$ bits, where ℓ is $\Theta(\log n)$ and $|I|$ is $\log^{\Theta(1)}(n)$ and $k = 2$.

Overall, the table will have $|I|^{\ell}$ such sum-check proofs, one for each choice of $r'_1, \ldots, r'_\ell$.

There are a few details to fix up, such as counting queries and random bits, but as we have described the proof so far, the size of the table is still at least $|I|^{\Theta(\ell)} = \log n^{\Theta(\log n)} = n^{\Theta(\log \log n)}$ which is not polynomial.

We can modify the proof in the following way so that the space required is polynomial. Encode the variable names in base h; so that rather than using $\{0,1\} \subseteq I \subseteq \mathbb{F}$, use $H \subseteq I \subseteq \mathbb{F}$ where $|H| = h = \log n$. In this way, one can reduce the number of field elements required to encode a variable or clause to $\ell' = O(\log n / \log \log n)$. This will have the advantage that $|I|^{\ell}$ will only be polynomial but it will have the drawback that instead of using multilinear extensions we will need to use extensions of maximum-degree $k = h - 1$. For example, it will mean that in the sum-check protocol the test will be that $\sum_{y \in H^\ell} p(y) = c_0$ and thus instead of checking that $f_i(0) + f_i(1) = c_{i-1}$ at each step, the verifier will need to check that $\sum_{j \in H} f_i(j) = c_{i-1}$.

The rest of the analysis including the total number of queries and random bits is sketched in the next lecture. □

Lecture 15

The PCP Theorem

May 25, 2004
Lecturer: Paul Beame
Notes: Ashish Sabharwal

At the end of last class we had nearly finished the proof of the following two theorems.

Theorem 15.1 (Babai-Fortnow-Levin-Szegedy). $\mathsf{NP} \subseteq \mathsf{PCP}(\mathrm{polylog}, \mathrm{polylog})$*; more precisely* $\mathsf{NP} = \mathsf{PCP}(\log n, \log^3 n)$.

Theorem 15.2 (Feige-Goldwasser-Lovasz-Safra-Szegedy). $\mathsf{NP} \subseteq \mathsf{PCP}(\log n \log\log n, \log n \log\log n)$.

Proof. Proof continued To review, in the last lecture, we first described PCP proofs that involved constructing a multilinear extension $\bar{a}$ of an assignment a and checking the arithmetized clauses $CC(w, a)$ for $w \in \{0,1\}^{3\log n+3} = \{0,1\}^{\ell}$ where each arithmetized clause had maximum degree 2 and total degree at most 4. We wanted to check that all these evaluated to 0. So, we used a Reed-Muller code that involved $r_1, r_2, \ldots, r_\ell \in_R I \subseteq \mathbb{F}$ and checked using the sum-check protocol that the polynomial sum $E_R(\bar{a}) = \sum_{w\in\{0,1\}^\ell} CC(w,a) \prod_{i,\, w_i=1} r_i = 0$. The sum-check protocol required a table of size $|I|^\ell \log|\mathbb{F}|$, the number of random bits used was $2\ell \log|I|$ and the number of queries was $\ell \log|\mathbb{F}|$. For the sum-check protocol to have a small failure probability we needed that $|I|$ is at least a constant factor larger than (#vars) $\cdot$ (maxdegree) $= 2\ell$.

In order to apply this test we need to ensure that the table $\bar{a}$ really is close to a multilinear function. To do this we used the aligned line test described in Lecture 13. From Corollary 13.3, for $|I| = O((\text{\#vars})^2 \cdot (\text{maxdegree})^2/\delta^2)$, the number of trials needed to show distance at most some small constant δ was $O((\#vars) \cdot (\text{maxdegree})/\delta) = O((\#vars) \cdot (\text{maxdegree}))$. Each trial required $O(\text{maxdegree})$ queries. The total number of queries is $O((\text{\#vars})(\text{maxdegree}^2)$. Plussing in the number of variables $\ell = \Theta(\log n)$ and the the maximum degree we test for which is 1 in the case of multilinearity we conclude that for $|I| = \Theta(\log^2 n)$, the total number of queries used in the multilinearity test is $O(\log n)$ each of which is an element of $\mathbb{F}$ which will require $O(\log|\mathbb{F}|) = O(\log\log n)$ bits. This is fine for what we wanted.

However, by Corollary 13.3, each trial uses (#vars) $\log|I|$ random bits so the total number of random bits used is $\Omega((\text{\#vars})^2(\text{maxdegree})\log|I|)$, which is $\Omega(\log^2 n \log\log n)$. This was more than our target of $O(\log n \log\log n)$ random bits. To fix this we can use pairwise independence in place of complete independence between the trials. We previously discussed this idea, due to Chor and Goldreich, in theory seminar in Winter Quarter.

Pairwise Independent Trials Suppose we have a test that uses r random bits per trial. Using pairwise independent hash functions $h_{a,b}(x) = ax + b$ over $\mathbb{F}_{2^r}$, we can do k' trials by choosing $a, b \in_R \mathbb{F}_{2^r}$ and

using $h_{a,b}(1), h_{a,b}(2), \ldots, h_{a,b}(k')$ for $k' < 2^r$ as pairwise independent random strings. This uses only $2r$ random bits overall. By Chebyshev's inequality, with high probability, the number of successful trials is close to the expected number when the trials are completely independent, for large enough k'.

In our case, $r =$ (#vars) $\log |I|$ random bits per trial are used and $k' = c$(#vars) $\cdot$ (maxdegree) trials is still sufficient with the additional errors introduced through the Chebyshev inequality. Thus a total of $O(\log n \log\log n)$ random bits suffice for the aligned line test of multilinearity. This is enough to yield the theorem that $\mathsf{NP} \subseteq \mathsf{PCP}(\log n \log\log n, \log n \log\log n)$.

In order to reduce the proof size to polynomial and the number of random bits to $O(\log n)$ toward the end of the last lecture we replaced the two element set $\{0,1\}$ with set H with $|H| = h = \log n$. We could now encode n variables with $\log n / \log\log n$ variables over the set $\{1, 2, \ldots, h\}$ by a simple change of basis. However, by this change, the maximum degree $k = 2$ in the sum-check protocol goes up to $O(\log n)$ since the assignment A is degree $h - 1$ and is no longer multilinear. Similarly, in the max-degree $h - 1$ test of the assignment A, the (maxdegree)^2 term which previously was constant now becomes significant and thus the total number of queries q grows to $O(\log^3 n)$ bits. The size of I also needs to grow to $\Theta(\log^4 n)$ to compensate for the growth in the max degree. However, using the pairwise independent trials the number of random bits from both the max-degree-k tests and the sum-check protocol are still $O($(#vars) $\log |I|)$ random bits which is now only $O(\log n)$ bits. Thus $\mathsf{NP} = \mathsf{PCP}(\log n, \log^3 n)$ follows. □

15.1 The PCP Theorem

The rest of this lecture will be devoted to the proof and implications of the following result:

Theorem 15.3 (PCP Theorem). $\mathsf{NP} = \mathsf{PCP}(\log n, 1)$

15.1.1 Implications on Hardness of Approximation

Before going into the proof of the PCP theorem, we give one example of what it implies for approximation problems. Let MAX3SAT be the problem of finding an assignment to a given 3CNF formula F that maximizes the number of clauses of F satisfied. An *approximation algorithm* for MAX3SAT with approximation factor γ finds an assignment that satisfies at least OPT/γ clauses, where OPT is the number of clauses satisfied by an optimal assignment.

MAX3SAT is a complete problem in the class MAXSNP of NP optimization problems introduced by Papadimitriou and Yannakakis. Each problem in this class has a constant factor polynomial time approximation algorithm but it is not known whether or not these approximation factors can be made arbitrarily close to 1. The following corollary of the PCP Theorem shows that this is unlikely in general.

Corollary 15.4. *There exists an $\epsilon > 0$ such that if there is a polytime $(1+\epsilon)$-approximation for* MAX3SAT *then* $\mathsf{P} = \mathsf{NP}$.

Proof. We will convert SAT into a 'gap' problem and show that if there is a good enough approximation to MAX3SAT, then SAT can be solved in polynomial time. The PCP theorem yields a polynomial time verifier $V^?_{\mathrm{SAT}}$ with the following behavior. Given a formula ψ and an assignment a, consider the polysize PCP proof $E(a)$ that a satisfies ψ. $V^?_{\mathrm{SAT}}$ looks at $E(a)$, uses $O(\log n)$ random bits, and makes $q = O(1)$ queries based on those bits. If $\psi \in \mathrm{SAT}$, then $V^?_{\mathrm{SAT}}$ accepts. If $\psi \notin \mathrm{SAT}$, then V_{SAT} accepts with probability at most $1/2$.

$V^?_{\text{SAT}}$ has $2^{O(\log n)}$ = polynomial number of random choices. Create one test circuit for each such choice r, $Test_r : \{0,1\}^q \to \{0,1\}$. The inputs to each of these test circuits are (different) $q = O(1)$ bits of $E(a)$. Convert each $Test_r$ into a circuit of size $O(2^q/q)$; this can be done because of Lupanov's bound on circuit size stated in Lecture 4. (A trivial bound of $q2^q$ is also sufficient since these are all constant size.) Now convert each circuit into a 3CNF by adding extra variables. This is again of size $O(2^q/q)$. Call the conjunction of all these 3CNF's ψ'. As described above, ψ' can be constructed from ψ in polynomial time.

If $\psi \in$ SAT, then $\psi' \in$ SAT. If $\psi \notin$ SAT, then for any assignment a, at least half the tests $Test_r$ are not satisfied and thus have at least one unsatisfied clause in each, implying that a total of at least a $q/(2 \cdot 2^q) = \Omega(q/2^q)$ fraction of clauses of ψ' are unsatisfied. Since $q = O(1)$, this is a constant fraction and we have a gap problem for CNF satisfiability. If MAX3SAT can be approximated within a constant factor, then this gap problem can be solved exactly, proving the Corollary. □

15.1.2 Outline of the PCP Theorem Proof

As shown in Lecture 12, it will be sufficient to prove that $\mathsf{NP} \subseteq \mathsf{PCP}(\log n, 1)$ because any proof on the right hand side can be trivially converted into an NP proof by enumerating the underlying tests for all possible $2^{O(\log n)} = n^{O(1)}$ random choices of the PCP verifier.

Variants of the Low Degree Test

We used the aligned line test for max-degree k in the PCP constructions above. Even with an improved analysis due to Arora and Safra, this test is not efficient enough for the PCP Theorem.

In the polynomial-size PCP construction above, the polynomials had max-degree $k = \log n$. Their total degree was not much larger, since the number of variables $\ell = \log n / \log\log n$ and thus their total degree $d = \log^2 n / \log\log n$. Let $P_{tot}(\ell, d)$ be the set of all total degree d polynomials in ℓ variables.

In the improvements of the above PCP constructions that yield the PCP theorem, an efficient total-degree test instead of an max-degree test is used. Since the original PCP Theorem there have been a number of total-degree tests proposed and the original analysis has been improved. These tests all work for the construction and provide different analysis complexity, exact PCP proof size and other parameter variations.

The original test, which is also easy to describe, is the Affine Line Test due to Arora, Lund, Motwani, Sudan, and Szegedy.

The Affine Line Test: To test a function $f : \mathbb{F}^\ell$

A. Choose $x, y \in_R \mathbb{F}^\ell$

B. Query the $d+1$ coefficients of $f_{x,y}(t)$ where we are supposed to have $f_{x,y}(t) = f(x+yt)$

C. Check that $f_{x,y}(t) = f(x+yt)$ holds at a random point $t \in_R \mathbb{F}$.

Note that $(x+yt)$ for different t's are uniformly spaced points along a line of slope y with x as the starting point. We state the following theorem without proof.

Theorem 15.5. *The affine line test has the following properties:*

A. If f has total degree at most d, then the test always accepts.

B. There is some $\delta_0 > 0$ such that if the test accepts with probability at least $1 - \delta$ for some $\delta < \delta_0$, then f is within a 2δ fraction of a total degree d polynomial, i.e. , $d(f, P_{tot}(\ell, d)) \leq 2\delta$.

Note that this theorem is much more efficient both in number of queries and random bits than the aligned line test. We now give a sketch of rest of the proof of the PCP theorem. Many of the details will be omitted for lack of time.

15.1.3 Three New Ideas

There are three additional key ideas that are used in the proof of the PCP theorem.

Self-correction. The original protocol in the last lecture examined $\bar{a}$ in only three random places. In general the proof will need a more flexible way to access the truth assignment $\bar{a}$ or other polynomials in the proof.

Suppose we know that $d(\bar{a}, P_{total}(\ell, d)) \leq \delta$. Let $\hat{a}$ be the closest point in $P_{total}(n, d)$. We would like to evaluate $\hat{a}$ instead of $\bar{a}$.

Claim 3. We can evaluate $\hat{a}$ at an arbitrary place and be correct with probability at least $1 - (d+1)\delta$.

Proof idea. To evaluate $\hat{a}(y)$, choose a random y and compute $\bar{a}(x+y), \bar{a}(x+2y), \ldots, \bar{a}(x+(d+1)y)$. Compute degree d polynomial $p(i) = \bar{a}(x + iy)$ by interpolation. Evaluate $\hat{a}(x) = p(0)$. The randomness used here is in selecting $y \in_R \mathbb{F}^\ell$. The number of random bits needed is therefore the same as before. The number of queries is $d + 1$. □

A Completely Different PCP.

Lemma 15.6 (Arora-Lund-Motwani-Sudan-Szegedy). $\mathsf{NP} \subseteq \mathsf{PCP}(n^2, 1)$

We will describe the main ideas behind this result in the next section.

Composition. [Arora-Safra] We know $\mathsf{NP} = \mathsf{PCP}(\log n, \text{polylog } n)$. Start with such a PCP verifier V. Given a PCP proof, V uses $O(\log n)$ random bits and checks $\log^c n$ places of the proof, for some constant $c \geq 0$. Here is where the idea of composition comes in – instead of checking all $n' = \log^c n$ bits of the proof, view this test as an NP-style check on n' bits. This NP-style test itself can be replaced by an "inner" PCP verifier with parameters $(n'^2, 1)$ using the completely different PCP above. The new PCP proof now contains the original "outer verifier" of type $\mathsf{PCP}(\log n, \text{polylog } n)$ and for each test on n' bits, an "inner verifier" of type $\mathsf{PCP}(n'^2, 1)$ that makes sure the test is satisfied. In fact, we will need to apply composition more than once.

For this composition to make sense, we also need to add a "consistency check" to make sure all assignments in the inner verifiers are consistent with a single outer verifier table. This part of the construction is quite hairy and uses very specific properties of the proof tables themselves. We will skip the details.

15.1.4 Outline of the Proof

Assume for now that all three of these new ideas work out. We get

$$\begin{aligned} \mathsf{NP} &\subseteq \mathsf{PCP}(\underbrace{\log n}_{\text{outer}} + \underbrace{\log^{2c} n}_{\text{inner}}, 1) \\ &= \mathsf{PCP}(\text{polylog } n, 1) \end{aligned}$$

Now do a composition again with this new verifier as the inner verifier. We get

$$\begin{aligned} \mathsf{NP} &\subseteq \mathsf{PCP}(\underbrace{\log n}_{\text{outer}} + \underbrace{\text{poly}(\log\log n)}_{\text{inner}}, 1) \\ &= \mathsf{PCP}(\log n, 1) \end{aligned}$$

15.1.5 Idea Behind $\mathsf{NP} \subseteq \mathsf{PCP}(n^2, 1)$

Let QUADRATIC-SAT be the following problem: given a family of polynomial equations of total degree at most 2 over $\mathbb{F}_2$, are they simultaneously satisfiable?

Theorem 15.7. QUADRATIC-SAT *is* NP*-complete.*

Proof idea. One can simulate CIRCUIT-SAT using this problem. The general idea is to use the quadratic equations to define the gates variables in the same way that such variables are used in conversion of CIRCUIT-SAT into 3SAT. For instance, an AND gate with inputs y_i and y_j, and output y_k translates into the total degree 2 equation $y_k = y_i \cdot y_j$ over $\mathbb{F}_2$. □

We will prove that QUADRATIC-SAT is in $\mathsf{PCP}(n^2, 1)$. Suppose we have an assignment a and polynomials $P_1, P_2, \ldots, P_m$ of total degree at most 2. The PCP verifier will work as follows.

A. Choose $r_1, r_2, \ldots, r_m \in \mathbb{F}_2$.

B. Check assignment a on the total degree at most 2 polynomial $P \equiv r_1P_1 + r_2P_2 + \ldots + r_mP_m$, *i.e.* , test whether $P(a) = 0$.

By our standard argument, if there is some i such that $P_i(a) \neq 0$ then the probability that $P(a) = 0$ is $1/2$.

The verifier will know the coefficients of P but they depend on the random choices of the r_i that are not know in advance so the proof table will need to give values for all possible quadratic polynomials. The proof table that the verifier will access to do this calculation corresponds to expressing a total degree 2 polynomial as:

$$P = b_0 + \sum_{i=1}^{n} b_i y_i + \sum_{i,j=1}^{n} c_{ij} y_i y_j$$

The PCP proof will include a table of $\sum_{i=1}^{n} b_i a_i$ for all possible $b \in \mathbb{F}_2^n$ as well as a table of $\sum_{i,j=1}^{n} c_{ij} a_i a_j$ for all possible $c \in \mathbb{F}_2^n$. The former of these is the well-known Hadamard code from coding theory, while the latter is what is called the Quadratic code.

To check that this is correct, the verifier needs to do the following three things:

Check linearity of the tables in b and c, resp. Let $f(b)$ denote the result of querying the first table on b. To check linearity, the verifier checks $f(b \oplus b') = f(b) \oplus f(b')$ for $b, b' \in_R \mathbb{F}_2^n$. This requires 3 queries and we state without proof that if the check succeeds with probability at least $1 - \delta$ then the function differs from a linear function in at most a δ fraction of entries. Linearity of the second table in c is verified similarly.

Self-correction. Because the specific query b needed for P might be at a place where the table is incorrect, so to compute $f(b)$, choose a $b' \in_R \mathbb{F}_2^n$ and instead compute $f(b) \leftarrow f(b \oplus b') \oplus f(b')$.

Consistency check between the two tables. Although the individual b and c tables may be individually linear there might be no relationship between them. For any $y \in \mathbb{F}^n$ and $b, b' \in_R \mathbb{F}_2^n$ observe that $(\sum_i b_i y_i)(\sum_j b'_j y_j) = \sum_{i,j} b_i b'_j y_i y_j$. Thus for consistency we need that $c_{ij} = b_i b'_j$. In particular we need to check that that $(\sum_i b_i a_i)(\sum_j b'_j a_j) = \sum_{i,j} b_i b'_j a_i a_j$. The first two summations can be evaluated directly from the first table, while the last summation can be evaluated using the second table with $c_{ij} = b_i b'_j$. Using self-correction for each of the three summation evaluations, this takes 6 queries.

The verifier uses $O(n^2)$ random bits overall to perform these checks and makes only a constant number of queries. This proves that $\mathsf{NP} \subseteq \mathsf{PCP}(n^2, 1)$.

Lecture 16

Circuit Lower Bounds for NP problems

May 27, 2004
Lecturer: Paul Beame
Notes: Nilesh Dalvi

In the few years after the definitions of NP-completeness, there was some hope that techniques such as diagonalization from recursive function theory would be able to resaolve the question. However, those hopes were dashed in the late 1970's by the following construction.

Theorem 16.1 (Baker-Gill-Solovay). *There exists oracles A, B such that*

$$\mathsf{P}^A = \mathsf{NP}^A, \mathsf{P}^B \neq \mathsf{NP}^B$$

Since diagonal arguments generally work even when the machines involved are given access to oracles, this theorem suggests that diagonalization cannot help in deciding if $\mathsf{P} = \mathsf{NP}$ or $\mathsf{P} \neq \mathsf{NP}$.

Throughout the 1970's, there was also more focus by Cook and others on approaching the P versus NP question via the following containments of complexity classes

$$\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP}.$$

This led to the use of more restrictive log-space reductions instead of polynomial-time reductions and to look at which problems could be solved in both polynomial time and polylogarithmic space with a view to separating classes such as L from NP. This led to the naming of the following classes of languages which eventually came to be named after Steve Cook.

Definition 16.1.

$$\begin{aligned} \mathsf{SC}^k &= \mathsf{TIMESPACE}(n^{O(1)}, \log^k n) \\ \mathsf{SC} &= \cup_k \mathsf{SC}^k \qquad \text{["Steve's Class" after Steve Cook].} \end{aligned}$$

Open Problem 16.1. Is $\mathsf{SC} = \mathsf{P}$? Is $\mathsf{NL} \subseteq \mathsf{SC}$?

In the late 1970's in part because of the proved weakness of diagonalization above, the study of non-uniform complexity in general and circuits in particular rose to prominence. In particular, both for complexity-theoretic reasons and for understanding the power of parallel computation, the following complexity class analogues of SC were suggested.

Definition 16.2.

$$\begin{aligned} \mathsf{NC}^k &= \mathsf{SIZEDEPTH}(n^{O(1)}, O(\log^k n)) \\ \mathsf{NC} &= \cup_k \mathsf{NC}^k \qquad \text{["Nick's Class" after Nick Pippenger].} \end{aligned}$$

If each gate has a constant time delay, problems solvable in NC can be solved in polylog time using a polynomial amount of hardware. Both to understanding how one would actually build such parallel machines it is natural to define uniform versions of the NC circuit classes, which express how easy it is to build the n-th circuit. There are many variants of such uniform complexity classes:

polytime uniform	: there is a TM that on input 1^n outputs the n^{th} circuit in time $n^{o(1)}$
log-space uniform	: there is a TM that on input 1^n outputs the n^{th} circuit using space $O(logn)$ or, equivalently, there is a TM that given a triple (u, v, op) of gate names u and v and an operation op determines whether or not u is an input to v and gate v is labeled by op and operates in linear space in the size of its input.
FO uniform	: the language (u, v, op) as above can be recognized by a first-order logic formula.

Theorem 16.2. *The following containment holds*

$$\textit{log-space uniform } \mathsf{NC}^1 \subseteq \mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{NC}^2$$

Proof sketch. log-space uniform $\mathsf{NC}^1 \subseteq \mathsf{L}$: An NC^1 circuit has $O(\log n)$ depth. A log-space machine can evaluate the circuit by doing a depth-first traversal using stack height at most $O(\log n)$ and accessing the gates as needed using the log-space constructibility of the circuit as needed. in log-space and, the circuit can be evaluated.

($\mathsf{NL} \subseteq \mathsf{NC}^2$) We show that directed graph reachability can be computed in NC^2. Graph reachability can be computed by using $\wedge - \vee$ matrix powering to compute transitive closure. This can be computed efficiently using repeated squaring.

$$A \to A^2 \to A^4 \to \cdots \to A^{2^{\log n}} = A^n$$

where A is the adjacency matrix. Each matrix squaring can be performed in $O(\log n)$ depth and polynomial size since there is a simple $O(\log n)$ depth fan-in circuit computing $\bigvee_{k=1}^{n}(a_{ik} \wedge a_{kj})$. Thus, graph reachability can be performed in $O(\log^2 n)$ depth and polynomial size. □

Open Problem 16.2. Is $\mathsf{NP} \not\subseteq \mathsf{NC}^1$? Even more specifically it is consistent with our current knowledge that $\mathsf{NP} \subseteq \mathsf{SIZEDEPTH}(O(n), O(\log n))$!

Additional Circuit Complexity Classes in NC

Definition 16.3. Define $\mathsf{AC}{-}\mathsf{SIZEDEPTH}(S(n), d(n))$ to be the circuit complexity class with appropriate size and depth bounds that allows unbounded fan-in $\vee$ and $\wedge$ gates in addition to binary fan-in $\vee$ and $\wedge$ gates. [The AC stands for "alternating class" or "alternating circuits".]
Define $\mathsf{AC}^k = \mathsf{AC}{-}\mathsf{SIZEDEPTH}(n^{O(1)}, O(\log^k n))$.

Analogously, we define $\mathsf{AC}[p]{-}\mathsf{SIZEDEPTH}(S(n), d(n))$ and $\mathsf{AC}^k[p]$ where one also allows unbounded fan-in $\oplus_p$ gates, where

$$\oplus_p(x_1, \cdots x_n) \begin{cases} 0 & \text{if } \sum x_i \equiv 0 \pmod p \\ 1 & \text{if } \sum x_i \not\equiv 0 \pmod p. \end{cases}$$

and $\mathsf{ACC}{-}\mathsf{SIZEDEPTH}(S(n), d(n))$ and ACC^k where where unbounded fan-in $\oplus_p$ gates for any values of p are allowed. [ACC stands for "alternating circuits with counters".

Finally, define threshold circuits $\mathsf{TC}{-}\mathsf{SIZEDEPTH}(S(n), d(n))$ and TC^k to allow threshold gates T_m^n, where

$$T_m^n(x_1, \cdots x_n) = \begin{cases} 1 & \text{if } \sum x_i \geq m \\ 0 & \text{otherwise.} \end{cases}$$

These circuits are important since TC^0 corresponds to bounded-depth neural networks.

Lemma 16.3. *Following containments hold*

$$\mathsf{NC}^0 \subseteq \mathsf{AC}^0 \subseteq \mathsf{AC}^0[p] \subseteq \mathsf{ACC}^0 \subseteq \mathsf{TC}^0 \subseteq \mathsf{NC}^1$$

Proof. All of the containments follow easily from definitions. For example $\mathsf{ACC}^0 \subseteq \mathsf{TC}^0$ because count can be implemented by threshold gates. □

Additionally, there is the following non-trivial containment:

Lemma 16.4. $\mathsf{NC}^1 \subseteq \mathsf{AC}{-}\mathsf{SIZEDEPTH}(n^{O(1)}, O(\log n/\log\log n))$.

Proof. Break the NC^1 circuit into $O(\log n/\log\log n)$ layers of depth $m = \log\log n$ each. Eac gate at the boundaries between the layers can be expressed as a binary fan-in circuit with at most 2^m inputs from the boundary gates of the previous layer. Any function on $M = 2^m$ inputs can be expressed as a (depth-2) DNF formula of size $M2^M = 2^m 2^{2^m} = O(n\log n)$ so we can replace the circuitry between each layer by the appropriate unbounded fan-in circuitry from these DNFs, retaining polynomial size but reducing depth by a factor of $\frac{1}{2}\log\log n$. □

The following are the two main theorems we will prove over the next lecture and a half. As stated, the latter theorem is stronger than the former but the proof techniques for the former yield sharper bounds and are interesting and useful in their own right.

Theorem 16.5 (Furst-Saxe-Sipser, Ajtai). *Parity, $\oplus_2$, is not in* AC^0.

Theorem 16.6 (Razborov, Smolensky). *Let $p \neq q$ be primes. Then $\oplus_p \notin \mathsf{AC}^0[q]$.*

Corollary 16.7. $\oplus_p \notin \mathsf{AC}^0[q]$ *where q is a prime power and p contains a prime factor not in q.*

For the rest of this lecture we give the proof of Theorem 16.5.

Intuition: For an unbounded fan-in $\vee$ gate, setting any bit to 1 fixes the output. In an unbounded fan-in $\wedge$ gate, setting any bit to 0 fixes the output. However, for a parity gate, all the inputs need to be fixed to determine the output. Therefore, set bits to simplify the AC^0 circuit (and eventually fix its value) while leaving some bits unset which ensure that the circuit cannot compute parity.

Definition 16.4. Define a *restriction* to be a function $\rho : \{1, \cdots n\} \to \{0, 1, *\}$, where

$$\rho(i) = \begin{cases} 0 & \text{means that variable } x_i \text{ is set to 0,} \\ 1 & \text{means that variable } x_i \text{ is set to 1, and} \\ * & \text{means that variable } x_i \text{ is not set.} \end{cases}$$

Let $*(\rho) = \rho^{-1}(*)$ denote the set of variables unset by ρ.
Define $f|_\rho$ or $C|_\rho$ as the simplification of the function or circuit that results from applying the restriction ρ.

Definition 16.5. Define $\mathcal{R}_p$ to be a probability distribution on the set of restrictions such that for each i, the probabilities of $\rho(i)$ being *, 0 and 1 are p, $\frac{1-p}{2}$ and $\frac{1-p}{2}$ respectively and are independent for each i.

Lemma 16.8 (Hastad's Switching Lemma). *Let $0 \leq p \leq 1$ and let F be an s-DNF formula, i.e., having terms of length at most s. For $\rho \in_R \mathcal{R}_p$,*

$$\Pr[F|_\rho \textit{ cannot be expressed as a } t\textit{-CNF}] < (5ps)^t.$$

The proof of this lemma is too long to present here but some useful intuition is in order. Suppose we examine the terms of F, one by one. Any clause that is not set to by ρ leaves an s-DNF remaining without that clause, which is a problem of essentially the same type as before. Given that the term is not set to 0 then every variable in the term is either unset or set according to its sign in the term. Given this, it has only a roughly $2p$ chance that it is unset versus set according to the term. Therefore the expected number of unset variables in any term is at most $2ps$ and it is very unlikely that more than one will be found in any term. Of course the above argument ignores all sorts of probability conditioning which yields the extra .

Corollary 16.9. *Let $f : \{0,1\}^n \to \{0,1\}$ be in* AC$-$SIZEDEPTH(S,d). *Then, there exists ρ such that $|*(\rho)| \geq n/(10^d(\log S+1)^{d-1})$ and $f|_\rho$ can be expressed as a $(\log S+1)$-DNF or CNF.*

Proof. Without loss of generality assume that all negations are at leaves and $\vee$ and $\wedge$ alternate.

The general idea is to apply the Hastad switching lemma to the subcircuits of the circuit computing f that are nearest the inputs (usually called the bottom level of the circuit). At the bottom level, the functions at the $\vee$ (resp. $\wedge$) gates are switched to $\wedge$ (reps. $\vee$) gates and merged with the level above.

In general, in applying the Hastad switching lemma, the argument will maintain $s = t = \log S + 1$ and set $p = \frac{1}{10(\log S+1)} = \frac{1}{10s}$. In this case

$$(5ps)^t = 2^{-t} = 2^{-\log S-1} = \frac{1}{2S}.$$

At the start of the argument however, the bottom level $\wedge$ or $\vee$ gates correspond to 1-DNF or 1-CNFs so one begins with $s = 1$ and $t = \log S+1$. In this first step $p = \frac{1}{10}$ is good enough to yield a $\frac{1}{2S}$ failure probability at most.

Let $i = 1$. For each gate g at the bottom level, the probability that g doesn't simplify under $\rho_i \in_R \mathcal{R}_p$ is less than $\frac{1}{2S}$. There are at most S such gates; so, the probability that there is some gate that doesn't simplify is less than $1/2$.

Note that $|*(\rho_i)|$ is a binomially distributed random variable with mean $E[|*(\rho_i)|] = pn$. Because when $p(1-p)n \to \infty$ the binomial distribution behaves in the limit like a normal distribution a constant fraction of its weight is above the mean, so we have $\Pr[|*(\rho_i)| \geq pn] \geq 1/3$.
Therefore $\Pr[\rho_i$ has $|*(\rho_i)| \geq pn$ and circuit depth shrinks by $1] \geq 1/6$. Hence, by probabilistic method there exists a ρ_i that has these properties. Fix it and repeat for $i+1$, reducing depth every time. This gives us a combined restriction ρ which is the composition of all the ρ_i and has the desired properties. □

Theorem 16.10. *Any* AC *circuit computing parity in size S and depth d has $S \geq 2^{\frac{1}{10}n^{1/(d-1)}} - 1$.*

Proof. To compute parity, we need

$$\begin{aligned} *(\rho) &\leq \log S + 1 \\ \Rightarrow \frac{n}{10^d(\log S+1)^{d-1}} &\leq \log S + 1 \\ \Rightarrow n &\leq 10^d(\log S+1)^d \\ \Rightarrow S+1 &\geq 2^{\frac{1}{10}n^{1/d}} \end{aligned}$$

To obtain a stronger result, observe that the subcircuits of depth $d-1$ that are combined to produce the parity function also require terms/clauses of size equal to the number of unset variables. Therefore we can apply the above argument to the depth $d-1$ subcircuits of the parity circuit and replace d by $d-1$. □

Note that for the size to be polynomial this requires depth $d = \Omega(\log n / \log \log n)$.

The above argument is essentially tight since parity can be computed by AC circuits of depth d and size $2^{O(n^{1/(d-1)})}$.

Lecture 17

Counting is hard for small depth circuits

June 1, 2004
Lecturer: Paul Beame
Notes: Sumit Sanghai

In this lecture we will give bounds on circuit size-depths which compute the function $\oplus_p$. More specifically we will show that a polynomial-sized constant depth $AC^0[q]$ circuit cannot compute $\oplus_p$.

Theorem 17.1 (Razborov,Smolensky). *Let $p \neq q$ be primes. Then $\oplus_p \notin AC^0[q]$.*

We will prove that $S = 2^{n^{\Omega(1/d)}}$ or $d = \Omega(\log n / \log\log S)$. Note that $AC^0[q]$ contains the operations $\wedge$, $\vee$, $\neg$ and $\oplus_q$ where $\oplus_q(x_1, \ldots, x_n) = \begin{cases} 0 & \text{if } \sum_i x_i \equiv 0 \pmod q \\ 1 & \text{otherwise.} \end{cases}$

To prove this theorem we will use the *method of approximation* introduced by Razborov.

Method of Approximation For each gate g in the circuit we will define a family A_g of allowable approximators for g. For the operation Op_g at gate g, we define an approximate version $\widetilde{Op_g}$ such that if $g = Op_g(h_1, \cdots, h_k)$ then $\widetilde{g} = \widetilde{Op_q}(\widetilde{h_1}, \cdots, \widetilde{h_k}) \in A_g$.

We will prove that there are approximators such that $\widetilde{Op}(\widetilde{h_1}, \cdots, \widetilde{h_k})$ and $Op(\widetilde{h_1}, \cdots, \widetilde{h_k})$ differ on only an ϵ-fraction of all inputs implying that the output $\widetilde{f} \in A_f$ differs from f on at most ϵS fraction of all inputs. We will then prove that any function in A_f differs from f on a large fraction of inputs proving that S is large given d.

Proof of Theorem 17.1. We will prove that $\oplus_2 \notin AC^0[q]$ where q is a prime greater than 2. The proof can be extended to replace $\oplus_2$ by any $\oplus_p$ with $p \neq q$.

The Approximators For a gate g of height d' in the circuit, the set of approximators A_g will be polynomials over $\mathbb{F}_q$. of total degree $\leq n^{\frac{d'}{2d}}$.

Gate approximators

- $\neg$ gates: If $g = \neg h$, define $\widetilde{g} = 1 - \widetilde{h}$. This yields no increase in error or degree.
- $\oplus_q$ gates: If $g = \oplus_q(h_1, \ldots, h_k)$, define $\widetilde{g} = (\sum_{i=1}^k \widetilde{h_i})^{q-1}$. Since q is a prime, by Fermat's little theorem we see that there is no error in the output. However, the degree increases by a factor of $q-1$.
- $\vee$ gate:
  Note that without loss of generality we can assume that other gates are $\vee$ gates: We can replace the

$\wedge$ gates by $\neg$ and $\vee$ gates and since the $\neg$ gates do not cause any error or increase in degree we can "ignore" them.
Suppose that $g = \bigvee_{i=1}^{k} h_i$. Choose $\bar{r_1}, \cdots, \bar{r_t} \in_R \{0,1\}^k$. Let $\widetilde{h} = (\widetilde{h_1}, \cdots, \widetilde{h_k})$. Then

$$\Pr[\bar{r_1} \cdot \widetilde{h} \equiv 0 \pmod q] = \begin{cases} 1 & \text{if } \bigvee i = 1^k \widetilde{h_i} = 0, \text{ and} \\ \leq 1/2 & \text{otherwise.} \end{cases}$$

(This follows because if $\bigvee_{i=1}^{k} \widetilde{h_i} = 1$ then there exists j such that $\widetilde{h_j} \neq 0$ in which case if we fix the remaining coordinates of $\bar{r_1}$, there is at most one choice for the j^{th} coordinate of $\bar{r_1}$ such that $\bar{r_1} \cdot \widetilde{h} \equiv 0 \pmod q$.)
Let $\widetilde{g_j} = (\bar{r_j} \cdot \widetilde{h})^{q-1}$ and define

$$\widetilde{g} = \widetilde{g_1} \vee \cdots \vee \widetilde{g_t} = 1 - \prod_{j=1}^{t} (1 - \widetilde{g_j}).$$

For each fixed vector of inputs $\widetilde{h}$,

$$\Pr[\, \widetilde{g} \neq \bigvee_{i=1}^{k} \widetilde{h_i} \,] \leq (1/2)^t.$$

Therefore, there exists $\bar{r_1}, \cdots, \bar{r_t}$ such that $\widetilde{g}$ and $\bigvee_{i=1}^{k} \widetilde{h_i}$ differ on at most a $(1/2)^t$ fraction of inputs.
Also note that the increase in degree from the $\widehat{h_i}$ to $\widehat{g}$ is $(q-1)t$. We will choose $t = n^{\frac{1}{2d}}/(q-1)$.

Thus we obtain the following lemma:

Lemma 17.2. *Let $q \geq 2$ be prime. Every $\mathsf{AC}[q]$ circuit of size S and depth d has a degree $((q-1)t)^d$ polynomial approximator over $\mathbb{F}_q$ with fractional error at most $2^{-t}S$.*
In particular, setting $t = \frac{n^{1/(2d)}}{q-1}$, there is a degree $\sqrt{n}$ approximator for the output of the circuit having error $\leq 2^{-\frac{n^{1/(2d)}}{q-1}} S$.

In contrast we have the following property of approximators for $\oplus_2$.

Lemma 17.3. *For $q > 2$ prime and $n \geq 100$, any $\sqrt{n}$ degree polynomial approximator for $\oplus_2$ over $\mathbb{F}_q$ has error at least $1/5$.*

Proof. Let $U = \{0,1\}^n$ be the set of all inputs. Let $G \subseteq U$ be the set of "good" inputs, those on which a degree $\sqrt{n}$ polynomial a agrees with $\oplus_2$.

Instead of viewing $\oplus_2$ as $\{0,1\}^n \to \{0,1\}$ we consider $\oplus_2' : \{-1,1\}^n \to \{-1,1\}$ where we interpret -1 as representing 1 and 1 as representing 0. In particular, $\oplus_2'(y_1, \cdots, y_n) = \prod_i y_i$. where $y_i = (-1)^{x_i}$. We get that $\oplus_2(x_1, \cdots, x_n) = 1$ if and only if $\oplus_2'(y_1, \cdots, y_n) = -1$.

We can see that the $x_i \to y_i$ map can be expressed using a linear map m as follows
$m(x_i) = 2x_i - 1$ and since q is odd, m has an inverse map $m^{-1}(y_i) = (y_i + 1)/2$

Thus, given a of $\sqrt{n}$-degree polynomial that approximates $\oplus_2$, we can get an approximator a' of $\sqrt{n}$ degree that approximates $\oplus_2'$ by defining

$$a'(y_1, \cdots, y_n) = m(a(m^{-1}(y_1), \cdots, m^{-1}(y_n))).$$

It is easy to see that a' and $\oplus_2'$ agree on the image $m(G)$ of G.

Let $\mathcal{F}_G$ be the set of all functions $f : m(G) \to \mathbb{F}_q$. It is immediate that

$$|\mathcal{F}_G| = q^{|G|}. \tag{17.1}$$

Given any $f \in \mathcal{F}_G$ we can extend f to a polynomial $p_f : \{1, -1\}^n \to F_q$ such that f and p_f agree everywhere on $m(G)$. Since $y_i^2 = 1$, we see that p_f is multilinear. We will convert p_f to a $(n + \sqrt{n})/2$-degree polynomial.

Each monomial $\prod_{i \in T} y_i$ of p_f is converted as follows:

- if $|T| \leq (n + \sqrt{n})/2$, leave the monomial unchanged.
- if $|T| > (n + \sqrt{n})/2$, replace $\prod_{i \in T} y_i$ by $a' \prod_{i \in \bar{T}} y_i$ where $\bar{T} = \{1, \ldots, n\} - T$. Since $y_i^2 = 1$ we have that $\prod_{i \in T} y_i \prod_{i \in T'} y_i = \prod_{i \in T \Delta T'} y_i$. Since on $m(G)$, $a'(y_1, \ldots, y_n) = \prod_{i=1}^n y_i$, we get that $\prod_{i \in T} y_i = a' \prod_{i \in \bar{T}} y_i$ on $m(G)$. The degree of the new polynomial is $|\bar{T}| + \sqrt{n} \leq (n - \sqrt{n})/2 + \sqrt{n} = (n + \sqrt{n})/2$.

Thus $|\mathcal{F}_G|$ is at most the number of polynomials over $\mathbb{F}_q$ of degree $\leq (n + \sqrt{n})/2$. Since each such polynomial has a coefficient over $\mathbb{F}_q$ for each monomial of degree at most $(n + \sqrt{n})/2$,

$$|\mathcal{F}_G| \leq q^M \tag{17.2}$$

where

$$M = \sum_{i=0}^{(n+\sqrt{n})/2} \binom{n}{i} \leq \frac{4}{5} 2^n \tag{17.3}$$

for $n \geq 100$. This latter bound follows from the fact that this sum consists of the binomial coefficients up to one standard deviation above the mean. In the limit as $n \to \infty$ this would approach the normal distribution and consist of roughly 68% of all weight. By n around 100 this yields at most 80% of all weight.

From equations 17.1,17.2 and 17.3 we get $|G| \leq |M| \leq \frac{4}{5} 2^n$. Hence the error $\geq 1/5$. □

Corollary 17.4. *For $q > 2$ prime, any $\mathsf{AC}^0[q]$ circuit of size S and depth d computing $\oplus_2$ requires $S \geq \frac{1}{5} 2^{\frac{n^{\frac{1}{2d}}}{q-1}}$*

Proof. Follows from Lemmas 17.2 and 17.3. □

This yields the proof of Theorem 17.1. □

From Corollary 17.4, we can see that for polynomial-size $AC[q]$ circuits computing $\oplus_2$, the depth $d = \Omega(\frac{\log n}{\log \log n})$. By the lemma from the last lecture that $\mathsf{NC}^1 \subseteq \mathsf{AC}{-}\mathsf{SIZEDEPTH}(n^{O(1)}, O(\frac{\log n}{\log \log n}))$ any asymptotically larger depth lower bound for any function would be prove that it is not in NC^1.

Our inability to extend the results above to the case that q is not a prime is made evident by the fact that following absurd possibility cannot be ruled out.

Open Problem 17.1. Is $NP \subseteq AC^0[6]$?

The strongest kind of separation result we know for any of the NC classes is the following result which only holds for the uniform version of ACC^0. It uses diagonalization.

Theorem 17.5 (Allender-Gore). $\mathrm{PERM} \notin \mathsf{UniformACC}^0$.