Last time: Polynomial-size Boolean circuits $P/poly$

Believe that $NP$ doesn't have poly-size circuits

**Karp-Lipton theorem**

If $NP \subseteq P/poly$, then $PH = \Sigma^P_2$

$P/poly$ also seems unlikely to contain $EXP$ (regardless of what we believe about Karp-Lipton)

**Meyer's theorem**

If $EXP \subseteq P/poly$, then $EXP = \Sigma^P_2$

Some care needed for the proof... Arm-Beck doesn't seem quite right. Hopefully the notes for lecture 9 are better.

**Corollary (of Meyer's theorem)**

If $P = NP$, then $EXP \neq P/poly$

Otherwise get contradiction to time hierarchy theorem

Upper bounds can yield lower bounds
Most functions are really hard

**Theorem (Shannon)**
A majority of functions \( f : \{0,1\}^n \rightarrow \{0,1\} \) require circuits of size \( \geq 2^n / (10n) \)

**Proof technique**
- Probabilistic method
- Union bound \( \Pr[U \land A_i] \leq \sum_i \Pr[A_i] \)

**TODAY**
- Some subclasses of P/poly (that we will talk more about later in the course)
- Randomized computation (Turing machines that can flip coins)
Massively parallel computing
(idealized model)

- lots of processors (say, \( n \))
- Fast communication network (communication between any pair in \( O(\log n) \) steps)
- Synchronized computation (global clock)
- Small amount of communication between each "clock ride" (operation on \( O(\log n) \) bits, say)

Say that problem has efficient parallel algorithm if instance of size \( n \)
can be solved on parallel computer with \( n^{O(1)} \) processors in time \( \log^{O(1)} n \)

Recall: \( \text{DEPTH of circuit} = \text{length of longest (directed) path from any input to output} \)
\( 0 \) is also OK

**DEF.** For \( d \in \mathbb{N}^* \), language \( L \) is in \( \text{NC}^d \) if decided by circuits \( \{ C \in \Sigma^{n \times n^{\log_d n}} : \text{poly}(n) \text{ size and depth } O(\log^d n) \} \)

\( \text{NC} = \bigcup_{d \in \mathbb{N}^*} \text{NC}^d \)
For the next definition, relax requirements on AND- and OR-gates so that they can have arbitrary fan-in. 0 also OK.

**DEF** For d ∈ \mathbb{N}^+, language \( L \) is in \( AC^d \) if decided by circuits \( \{ C_n \}_{n \in \mathbb{N}} \) with unbounded fan-in AND/OR-gates (and many NOT-gates) of poly(n) size and depth \( O(\log \log n) \).

\[ NC^0 \text{ not so interesting} \]

Constant depth - dependence on constant # of gates already \( AC^0 \) interesting

Fan-in for \( AC^d \) at most poly(n) (why?)

So unbounded fan-in can be simulated in log depth

\[ AC^0 \subseteq NC^1 \subseteq AC^1 \subseteq NC^2 \subseteq \ldots \]

This containment is known to be strict! (Yay!) Will prove it later in the course.
THEOREM

Language \( L \) has efficient parallel algorithm iff \( L \in \text{NC} \)

Note: Algorithm is uniform if circuit is uniform.
Non-uniform circuit \( \Rightarrow \) Algorithm with advice.

Proof sketch:

\((\Rightarrow)\) Suppose \( N \) processors, time \( D \)

Build \( D \) layers of \( N \) subcircuits each.
Circuit \( i \) in layer \( d \) does computation of processor \( i \) at time step \( d \).
Communication network - circuit wires between subcircuits.

\((\Leftarrow)\) Suppose \( L \in \text{NC} \) decided by \( \{C_n\}_{n \in \mathbb{N}} \). Let parallel computer read description of \( C_n \).

Now let every processor take responsibility for simulating a gate.
Send output to processors similarly gates that use this value.
Is it possible for every problem in \( \text{P} \) to find an efficient parallel implementation?

In many cases: yes! (addition, multiplication, division, matrix determinant, matrix rank, matrix inverse, etc.)

But always? Probably no.

What are the hardest problems in \( \text{P} \)?

**Definition** Language \( L \) is \( \text{P} \)-complete if

1. \( L \in \text{P} \)
2. \( \forall L' \in \text{P} \) it holds that \( L' \) is logspace-reducible to \( L \)

If \( L \) \( \text{P} \)-complete and \( L \in \text{NC} \), then \( \text{P} = \text{NC} \)

\[
\text{CIRCUIT EVAL} = \left\{ \langle C, x \rangle \mid x \in \{0,1\}^n \right. \\
\left. C \text{ is a } n \text{-input circuit} \right\}
\]

**Theorem** Circuit Eval is \( \text{P} \)-complete.
RANDOMIZED COMPUTATION

- Randomness as a computational resource
- Lots of deep & fascinating questions here — see Ch 8 in Aho-Berlekamp
- We'll get straight to the point: study Turing machines that can flip fair random coins

**DEF:** PROBABILISTIC TURING MACHINE (PTM)

A Turing machine with two transition functions $\delta_0$, $\delta_1$.
In each step, apply
$\delta_0$ with probability $\frac{1}{2}$
$\delta_1$ with probability $\frac{1}{2}$

Output of $M$ on $x$, $M(x)$, now a random variable

PTM runs in time $T(n)$ if $\forall x$ halts in $\leq T(|x|)$ steps regardless of random choices

What should it mean that such a machine decides a language?
Compare to nondeterminism

- NDTM accepts if $\exists$ one (out of exponentially many) accepting branch
- PTM: look at fraction of accepting branches
For language $L \subseteq \{0,1\}^*$ and $x \in \{0,1\}^*$, define
\[
L(x) = \begin{cases} 
1 & \text{if } x \in L \\
0 & \text{if } x \notin L 
\end{cases}
\]

Our model for efficient probabilistic/randomized computation:

\[\text{BPP} \text{ bounded-error probabilistic polynomial time}\]

**DEF 2.** A PTM $M$ decides $L$ in time $T(n)$ if $\forall x$

$M$ halts in $T(1|x|)$ steps and $\Pr \left[ M(x) = L(x) \right] \geq 2/3$.

\[\text{BPTIME}(T(n)) = \text{languages decided by PTMs in } O(T(n)) \text{ time}\]

\[\text{BPP} = \bigcup_{c \in \mathbb{N}} \text{BPTIME} (n^c)\]

Practically over random choices, not over inputs.

Constant 2/3 arbitrary (will see later).

Don't need perfectly fair coins (but we'll ignore this).

**PROP 3.** $L \in \text{BPP}$ if exist poly-time (deterministic) TM $M$ and polynomial $p$ s.t. for every $x$

\[\Pr_{r \in \{0,1\}^p(1|x|)} \left[ M(x,r) = L(x) \right] \geq 2/3\]

Notational aside:

Uniform sampling from $\{0,1\}^n$:

$x \in \{0,1\}^n$

$x \sim \{0,1\}^n$

$x \sim \text{Un}$

**COR 4.** $P \subseteq \text{BPP} \subseteq \text{EXP}$

**Proof** Can try all possible random strings in exponential time and compute success probability.
Can't prove even $\text{BPP} \neq \text{NEXP}$

What about $\text{BPP} \text{ vs } \text{ P}$?

Fairly strong reasons to believe $\text{P} = \text{BPP}$!

[Discussed in Chs 19-20 in Arora-Barak—may not have time to cover this.]

Example of the power of randomization

**POLYNOMIAL IDENTITY TESTING**

given: polynomial (multivariate) with integer coeff. In implicit form

decide: Is the polynomial identically zero?

representation: algebraic circuit

like Boolean circuits, but gates are $+$, $-$, $\times$
can also have constants $0,1,...$ if we wish
inputs $x_1, ..., x_n$
single output node (sign)

not hard to see: computes some polynomial

$\text{ZEROP} = \{ \text{algebraic circuits corresponding to } \}$

polynomials that are identically zero

why identity testing?

given $C, C'$, construct $D = C - C'$
and check if $D \in \text{ZEROP}$

Note compact representation

$\prod_{i=1}^{n} (1+ x_i)$ has $2^n$ terms

Circuit of size $2n$
SCHWARTZ-ZIPPEL LEMMA

Let \( p(x_1, x_2, \ldots, x_m) \) be non-zero poly of total degree \( \leq d \). Let \( S \) finite set of integers.
Then for \( a_1, \ldots, a_m \) chosen from \( S \) uniformly randomly with replacements
\[
P_0 \left[ p(a_1, \ldots, a_m) = 0 \right] \geq 1 - \frac{d}{|S|}
\]

Proof: Induction over \( m \).

Base case \( m = 1 \): Univariate polynomial
Degree \( \leq d \) \( \Rightarrow \) at most \( d \) roots.
So \( p \) can evaluate to zero on at most \( d \) out of \( |S| \) integers.
Inductive step: See Ahoa-Sarkar App A.6

TESTING IDEA

Circuit of size \( m \) \( \Rightarrow \leq m \) multiplications
\( \Rightarrow \) degree \( \leq 2^m \)
So pick \( a_1, \ldots, a_m \in [1, 10 \cdot 2^m] \), evaluate circuit, and apply Schwartz-Zippel
If circuit \( C \) encodes zero poly \( \Rightarrow \) result always 0
if non-zero poly \( \Rightarrow \) 90\% of non-zero output.

Problem: If degree \( \times 2^m \), then numbers grow as large as \( (10 \cdot 2^m)^2 \) exponentially many bits.
Hard to do in poly time...

Solution: "fingerprinting" compute modulo \( k \leq [2^m] \)