CMPSCI 601: **Recall From Last Time** Lecture 26

Theorem: All CFL's are in $\mathbf{sAC}^1$.

Facts: ITADD, MULT, ITMULT and DIVISION on n-bit integers are all in $\mathbf{ThC}^0$.

Th The following problems are complete for $\mathbf{PSPACE} = \mathbf{NPSPACE} = \mathbf{ATIME}[n^{O(1)}]$:

QSAT, GEOGRAPHY, SUCCINCT REACH.

A *permutation* of a finite set S is a one-to-one onto function from S to itself, The *composition* of two permutations is the permutation we get by performing first one and then the next. Composition is not commutative – the set of all permutations of five elements form the *non-commutative group* S_5 with composition as the operation.

The S_5 iterated multiplication problem (S_5-MULT) is to input n elements of S_5 (in order) and determine their composition. (As a language, it is the set of pairs (w, t) such that $w \in S_5^*$, $t \in S_5$, and w multiplies to t.) Clearly a DFA can carry out the sequence of multiplications, so S_5-MULT is a *regular* language.

Theorem 26.1 *S_5-MULT is complete for* $\mathbf{NC}^1$. *Specifically, if C is an n-input circuit of depth d and fan-in two, we can take a string x of length n and construct a sequence of 4^d permutations that multiplies to a non-identity permutation iff $C(x) = 1$.*

Notation: A *five-cycle* $(abcde)$ is the permutation that takes a to b, b to c, c to d, d to e, and e to a, where {a,b,c,d,e} = {1,2,3,4,5}.

Lemma: There exist five-cycles σ and τ such that $\sigma\tau\sigma^{-1}\tau^{-1}$ is a five-cycle. (This permutation is called the *commutator* of σ and τ.)

Proof: $(12345)(13542)(54321)(24531) = (13254)$.

Fact: (basic group theory) If α and β are both five-cycles, then $\alpha = \gamma\beta\gamma^{-1}$ for some permutation γ.

Proof: (of Barrington's Theorem) Use induction on the depth d of the circuit. For each gate g we'll construct a sequence $s(g)$ such that $s(g)$ evaluates to the five-cycle (12345) if g evaluates to 1 and $s(g)$ evaluates to the identity otherwise. By the Fact, if we can get one five-cycle we can get any other with a sequence of the same length.

Base Case: $d = 0$ and the gate is an input. Look up the literal and let $s(g)$ consist of one permutation, (12345) if the literal is true and the identity if it is false.

NOT Gates: If h is the NOT of g, compose $s(g)$ with (54321). This gives the identity if g is true and (54321) if g is false. Using the Fact, normalize to give (12345) if h is true and the identity if h is false.

AND Gates: Suppose h is the AND of g_1 and g_2 and each of g_1 and g_2 have depth d. Using $s(g_1)$ and $s(g_2)$, we construct four sequences of length 4^d each:

- a_1 yields (12345) if g_1 is true and the identity otherwise,
- a_2 yields (13542) if g_2 is true and the identity otherwise,
- b_1 yields (54321) if g_1 is true and the identity otherwise, and
- b_2 yields (24531) if g_2 is true and the identity otherwise.

Calculation: $a_1a_2b_1b_2$ yields (13254) if g_1 and g_2 are both true, and the identity otherwise.

Conclusion: If C is a depth $O(\log n)$ circuit, we get a sequence of length $4^{O(\log n)}$, which is polynomial. We have reduced the circuit evaluation problem to an S_5-MULT instance that is only polynomial size. ♠

An Application to PSPACE

Fact: **PSPACE** is characterized by circuits of polynomial *depth*.

Corollary: Any **PSPACE** problem can be reduced to an instance of S_5-MULT of length $2^{n^{O(1)}}$.

Corollary: (Cai-Furst) Any **PSPACE** problem can be solved by a log-space Turing machine that:

- has access to a read-only clock
- wipes its entire working memory every poly-many steps, except for *three safe bits*.

We see in an algorithms course that it is sometimes to our advantage to use randomness in solving a problem. For example, Quicksort has good average-case but bad worst-case behavior. Flipping our own coins can (with high probability) keep us out of the bad cases.

In a competitive situation we may be worried about our opponent predicting our move. Flipping our own coins may make this impossible and guarantee us some minimum level of expected success.

Random sampling of a large space may give us a good idea of the results of an impractically large exhaustive search. There is a large body of mathematics telling us what inferences we may reliably make from such sampling.

Is randomness a powerful tool in general? In complexity theory we attack this question by looking at problems where randomization seems practical, and comparing classes of such problems to deterministic and nondeterministic classes.

In a moment, we'll look at *primality testing*, which until recently *was* the most famous example of a problem that could be solved in polynomial time with high probability by a randomized algorithm, but which was not known to be in ¶. This example has been taken from us, however, by Agarwal, Saxena, and Kayal, who in 2002 gave a deterministic algorithm to test a number for primality in polynomial time.

[P] gives two other interesting randomized algorithms:

- Testing whether a matrix of polynomials has determinant identically zero, and
- Using a random walk through the space of settings to find a satisfying instance of a 2-CNF formula

$$\text{PRIME} \quad = \quad \{m \in \mathbf{N} \mid m \text{ is prime}\}$$

Proposition 26.2 $\overline{\text{PRIME}} \in \mathbf{NP}$

Proof:

$$m \in \overline{\text{PRIME}} \Leftrightarrow m < 2 \quad \vee$$
$$(\exists xy)((1 < x, y < m) \wedge x \cdot y = m)$$

♠

Question: Is PRIME $\in$ **NP**?

Fact 26.3 (Fermat's Little Theorem): *Let p be prime and $0 < a < p$, then,*

$$a^{p-1} \equiv 1 \,(\mathrm{mod}\, p)$$

$$\mathbf{Z}_n^\star \quad = \quad \{a \in \{1, 2, \ldots, n-1\} \mid \mathrm{GCD}(a, n) = 1\}$$

$Z_n^\star$ is the multiplicative group of integers mod n that are relatively prime to n.

Euler's Phi: $\varphi(n) \quad = \quad |\mathbf{Z}_n^\star|$

Proposition 26.4 *If $n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}$ is the prime factorizaton of n, then*

$$\varphi(n) \quad = \quad n(p_1 - 1)(p_2 - 1) \cdots (p_k - 1)/(p_1 p_2 \cdots p_k)$$

Theorem 26.5 *(***Euler's Theorem):** *For any n and any $a \in \mathbf{Z}_n^\star$,*

$$a^{\varphi(n)} \equiv 1 \,(\mathrm{mod}\, n)$$

Fact 26.6 *Let $p > 2$ be prime. Then $\mathbf{Z}_p^\star$ is a cyclic group of order $p-1$. That is,*

$$\mathbf{Z}_p^\star \quad = \quad \{a, a^2, a^3, \ldots, a^{p-1}\}$$

$$m \in \text{PRIME} \quad \Leftrightarrow \quad (\exists a \in \mathbf{Z}_m^\star)(\text{ord}(a) = m-1)$$

Theorem 26.7 *[Pratt]*
PRIME $\in$ **NP**.

Proof:
Given m,

1. Guess a, $1 < a < m$
2. Check $a^{m-1} \equiv 1 \pmod{m}$ by repeated squaring.
3. Guess prime factorization,
$$m - 1 \;=\; p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}$$
4. Check for $1 \leq i \leq k$,
$$a^{m-1/p_i} \not\equiv 1 (\text{mod } m)$$
5. Recursively check that $p_1, p_2, \ldots, p_k$ are prime.

$$T(n) \quad = \quad O(n^2) + T(n-1)$$

$$T(n) \quad = \quad O(n^3) \qquad \spadesuit$$

Corollary 26.8 *PRIME and Factoring are in* **NP**$\cap$*co-***NP**.

Let $a \in \mathbf{Z}_m^\star$

a is a *quadratic residue* mod m iff,

$$(\exists b)(b^2 \equiv a \,(\mathrm{mod}\, m))$$

For p prime let,

$$\left(\frac{a}{p}\right) = \begin{cases} 1 & \text{if } a \text{ is a quadratic residue mod } p \\ -1 & \text{otherwise} \end{cases}$$

Generalize to $\left(\frac{a}{m}\right)$ when m is not prime,

$$\left(\frac{a}{mn}\right) = \left(\frac{a}{m}\right)\left(\frac{a}{n}\right)$$

Fact 26.9 [Gauss]**(Quadratic Reciprocity)** *For odd* a, m*,*

$$\left(\frac{a}{m}\right) = \begin{cases} \left(\frac{m}{a}\right) & \textit{if } a \equiv 1 \pmod 4 \textit{ or } m \equiv 1 \pmod 4 \\ -\left(\frac{m}{a}\right) & \textit{if } a \equiv 3 \pmod 4 \textit{ and } m \equiv 3 \pmod 4 \end{cases}$$

$$\left(\frac{2}{m}\right) = \begin{cases} 1 & \textit{if } m \equiv 1 \pmod 8 \textit{ or } m \equiv 7 \pmod 8 \\ -1 & \textit{if } m \equiv 3 \pmod 8 \textit{ or } m \equiv 5 \pmod 8 \end{cases}$$

This 200-year-old result gives us an efficient way to calculate $\left(\frac{a}{m}\right)$, that uses time polynomial in the *length* of m.

Fact 26.10 [Gauss] *For p prime, $a \in \mathbf{Z}_p^\star$,*

$$\left(\frac{a}{p}\right) \quad \equiv \quad a^{\frac{p-1}{2}} \pmod{p}$$

Fact 26.11 *If m is not prime then,*

$$\left|\{a \in \mathbf{Z}_m^\star \mid \left(\frac{a}{m}\right) \equiv a^{\frac{m-1}{2}} (\text{mod } m)\}\right| < \frac{m-1}{2}$$

Solovay-Strassen Primality Algorithm:

1. Input is odd number m
2. For $i := 1$ to k **do** $\{$
3. choose $a < m$ at random
4. **if** GCD$(a, m) \neq 1$ **return**("not prime")
5. **if** $\left(\frac{a}{m}\right) \not\equiv a^{\frac{m-1}{2}} (\text{mod } m)$ **return**("not prime")
6. $\}$
7. **return**("probably prime")

Theorem 26.12 *Suppose we run Solovay-Strassen on a number m. Then:*

- *If m is prime then the answer is always "probably prime".*
- *If m is not prime, then the probability of "probably prime" is less than $1/2^k$.*

Corollary 26.13 PRIME $\in$ *"Truly Feasible"*

The new Agarwal-Saxena-Kayal algorithm is a great theoretical achievement (solving an open problem dating back to at least 1970) but is *not* the best way in practice to test primality. Solovay-Strassen (or the related Miller-Rabin algorithm) runs faster and gives you numbers that are reasonably certain to be prime. If you want a prime of a given size, you generate random numbers until you get one that passes the test many times. There are enough primes so that you can expect this not to take too long.

FACTORING is still believed to be hard for conventional computation. But – there is a 1994 algorithm due to Shor that solves FACTORING in poly time on a *quantum computer*. Only very small quantum computers have so far been built, but one of them has successfully factored the number 15 using something like Shor's method.

What does it mean for a problem to be solvable in "random polynomial time"? We can define a probabilistic TM easily enough by having an NDTM M flip a coin for each of its classes. There are then *four* different poly-time complexity classes defined in the literature!

We define Prob(M, x) to be the probability that M accepts x. Remember that A is in **NP** if there exists M such that Prob$(M, x) > 0$ iff $x \in A$. The new classes have similar definitions:

- A is in **RP** if there exists M such that Prob$(M, x) \geq 1/2$ for all $x \in A$ and Prob$(M, x) = 0$ for all $x \notin A$.
- A is in **ZPP** if both A and $\overline{A}$ are in **RP**.
- A is in **BPP** if there exists M such that Prob$(M, x) > 2/3$ for all $x \in A$ and Prob$(M, x) < 1/3$ for all $x \notin A$.
- A is in **PP** if there exists M such that Prob$(M, x) > 1/2$ iff $x \in A$.

Which of these classes are practical? After all, you don't *want* to put up with a significant probability of a wrong answer.

RP, the class generalizing the Solovay-Strassen primality algorithm, is pretty good. As we've seen, repeated independent trials can reduce our error probability to exponentially small.

ZPP is even better in one sense, because if we try *both* **RP** algorithms repeatedly, in a *constant expected number of trials* we will get a *guaranteed answer*. This is historically called a "Las Vegas" algorithm (provably correct, probably fast) as opposed to an **RP** or **BPP** "Monte Carlo" algorithm (probably correct, provably fast).

PP, on the other hand, is *completely useless* in practice. If the probability of acceptance is very very close to $1/2$, the number of trials needed to make a statistical prediction of whether it is over or under 1/2 could be exponential.

But for **BPP** we are in good shape!

Proposition 26.14 *If $S \in$* **BPP** *then there is a probabilistic, polynomial-time algorithm A' such that for all n and all inputs w of length n,*

$$\textbf{if } (w \in S) \textbf{ then } \text{Prob}(A'(w) = 1) \geq 1 - \frac{1}{2^n}$$

$$\textbf{if } (w \notin S) \textbf{ then } \text{Prob}(A'(w) = 1) \leq \frac{1}{2^n}$$

Proof: Iterate A polynomially many times and answer with the majority. The probability the mean is off by $\frac{1}{3}$ decreases exponentially with n — the formal proof uses Chernoff bounds. ♠

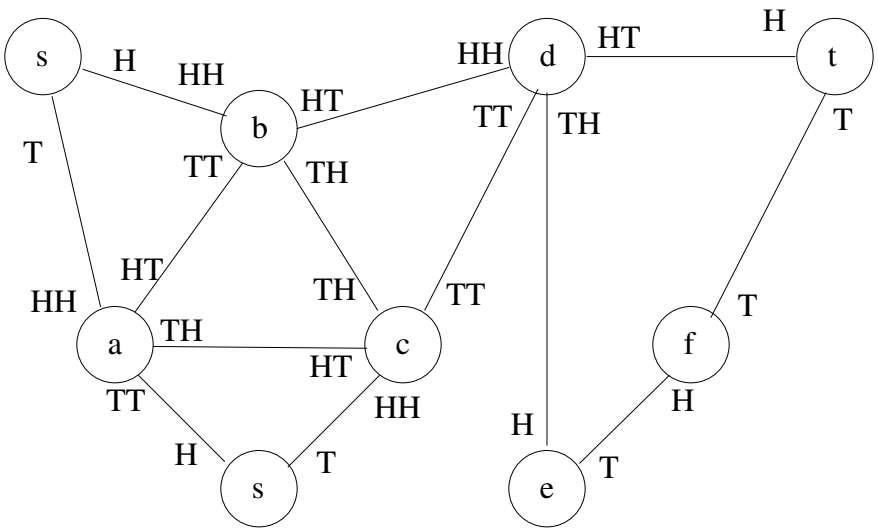
Is **BPP** equal to **P**???

Probably, because pseudo-random number generators are good.

It's *probably* possible to build a poly-time deterministic generator that gives numbers that are *indistinguishable from random* to any poly-time procedure. Such a generator would allow us to *derandomize* **BPP**.

It's not hard to show that if a language is in **BPP**, it has *non-uniform* poly-size circuits, i.e., it is in the non-uniform class **PSIZE**. This is because if we make the probability small enough that a random string causes the **BPP** algorithm to be wrong on a given input, we can ensure that some string exists that is right on *all inputs*. There *exists* a circuit that has this string hard-wired into it.

$$\text{REACH}_u \quad = \quad \{G, \text{ undirected} \mid s \overset{\star}{\underset{G}{\rightarrow}} t\}$$

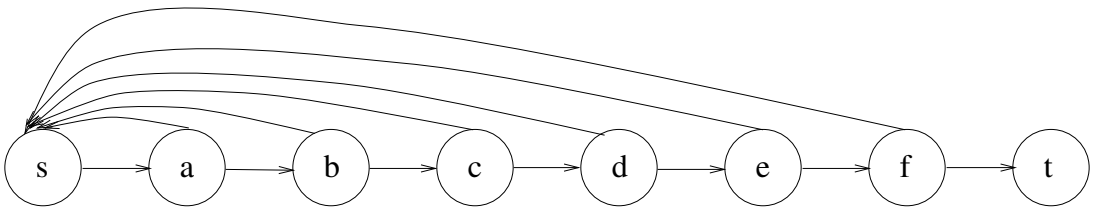


Fact 26.15 *Let $T(i)$ be the expected number of steps in a random walk to visit all vertices in connected graph G, starting from i. Then,*

$$T(i) \ \leq \ 2e(n-1)$$

Corollary 26.16

$$\text{REACH}_u \ \in \ \text{BPL}$$



A look at this *directed* graph should convince you that a random walk on it is *not* likely to reach all vertices in polynomial time. To get to vertex t from s you would have to guess right about n times in a row.

It's very plausible that REACH_u is in $\mathbf{L}$, and one might hope to prove it by derandomizing the random walk. (There must exist a single sequence of choices of size $O(n^3)$ that visits every node of *any* undirected labelled n-node graph.) But randomization doesn't seem to help much with the general REACH problem.

[Goldwasser, Micali, Rackoff], [Babai]

Decision problem: D; input string: x

Two players:

Prover — Merlin is computationally all-powerful. Wants to convince **Verifier** that $x \in D$.

Verifier — Arthur: probabilistic polynomial-time TM. Wants to know the truth about whether $x \in D$.

Input = x; $\quad n = |x|$; $\quad t = n^{O(1)}$

0.	**A** has x	**M** has x
1.	flip σ_1, compute m_1 $\longrightarrow$	
2.		$\longleftarrow m_2$
3.	flip σ_3, compute m_3 $\longrightarrow$	
4.		$\longleftarrow m_4$
$\vdots$	$\vdots$	$\vdots$
$2t$.		$\longleftarrow m_{2t}$
$2t+1$.	flip σ_{2t+1}, accept or reject	

Definition 26.17 $D \in$ IP iff there is such a polynomial-time interactive protocol

1. If $x \in D$, then there exists a strategy for $\mathbf{M}$

$$\text{Prob}\{\mathbf{A} \text{ accepts}\} > \frac{2}{3}$$

2. If $x \notin D$, then for all strategies for $\mathbf{M}$

$$\text{Prob}\{\mathbf{A} \text{ accepts}\} < \frac{1}{3}$$

♠

Observation 26.18 *Iterating makes probabilities of error exponentially small.*

Special Cases of IP**:**

- Deterministic Arthur = **NP**
- No Merlin = **BPP**

Graph Non-Isomorphism $\in$ AM

Input = $G_0, G_1, n = \|G_0\| = \|G_1\|$

0.	**A** has G_0, G_1	**M** has G_0, G_1
1.	flip $\kappa : \{1, \ldots, r\} \to \{0, 1\}$ flip $\pi_1, \ldots, \pi_r \in S_n$ $\pi_1(G_{\kappa(1)}), \ldots, \pi_r(G_{\kappa(r)}) \longrightarrow$	
2.		$\longleftarrow m_2 \in \{0, 1\}^r$
3.	accept iff $\kappa = m_2$	

Proposition 26.19 *Graph Non-Isomorphism* $\in$ AM

Proof: If $G_0 \ncong G_1$, then **A** will accept with probability 1.

If $G_0 \cong G_1$, then **A** will accept with probability $\leq 2^{-r}$. ♠

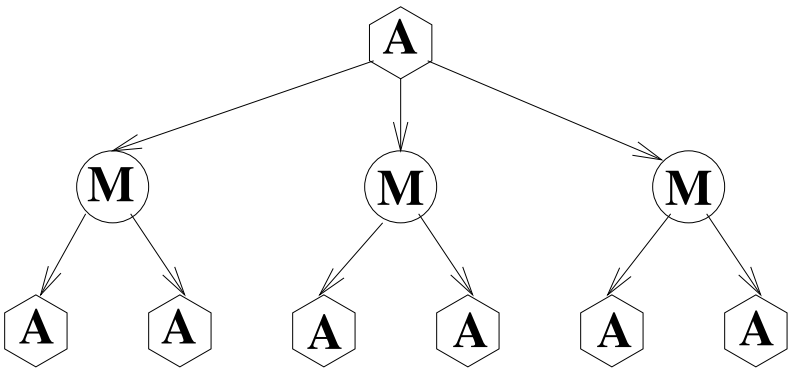
Corollary 26.20 *If Graph Isomorphism is* **NP***-complete then* PH *collapses to* Σ_2^p.

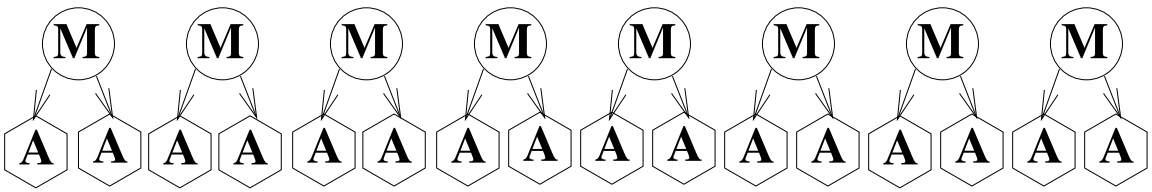
Fact 26.21 Shamir's Theorem: IP $=$ **PSPACE**

proof that IP $\subseteq$ **PSPACE:** Evaluate the game tree.

For M's moves choose the maximum value.

For A's moves choose the average value.





Hard Direction: Construct an interactive proof that a string is in QSAT. There are proofs in [P] and in Sipser.

Any decision problem $D \in$ **NP** has a deterministic, polynomial-time verifier.

By adding randomness to the verifier, we can greatly restrict its computational power and the number of bits of Π that it needs to look at, while still enabling it to accept all of **NP**.

We say that a verifier **A** is $(r(n), q(n))$-*restricted* iff for all inputs of size n, and all proofs Π, **A** uses at most $O(r(n))$ random bits and examines at most $O(q(n))$ bits of its proof, Π.

Let PCP$(r(n), q(n))$ be the set of boolean queries that are accepted by $(r(n), q(n))$-restricted verifiers.

Fact 26.22 (PCP Theorem) **NP** $=$ PCP$[\log n, 1]$

The proof of this theorem is pretty messy, certainly more than we can deal with here. But we can look at the applications of the PCP Theorem to approximation problems.

MAX-3-SAT: Given a 3CNF formula, find a truth assignment that maximizes the number of true clauses.

$$(x_1 \vee x_2 \vee \overline{x_3}) \wedge (x_1 \vee x_4 \vee \overline{x_5}) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_4}) \wedge (x_2 \vee \overline{x_3} \vee \overline{x_4})$$
$$\wedge (\overline{x_2} \vee x_3 \vee x_5) \wedge (\overline{x_3} \vee \overline{x_4} \vee \overline{x_5}) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_3) \wedge (\overline{x_2} \vee \overline{x_4} \vee x_5)$$

Proposition 26.23 MAX-3-SAT *has a polynomial-time* $\epsilon = \frac{1}{2}$ *approximation algorithm.*

Proof: Be greedy, set each variable in turn to the better value. ♠

You can do better – a random assignment gets 7/8 of the clauses.

Open for Years: Assuming **NP** $\neq$ **P** is there some ϵ, $0 < \epsilon < 1$ s.t. MAX-3-SAT has no PTIME ϵ-approximation algorithm?

Theorem 26.24 *The PCP theorem (***NP** $=$ PCP$[\log n, 1]$*)*

is equivalent to the fact that

If **P** $\neq$ **NP**, *then*

For some ϵ, $1 > \epsilon > 0$,

MAX-3-SAT *has no polynomial-time,* ϵ*-approximation algorithm.*

Fact 26.25 MAX-3-SAT *has a PTIME approximation algorithm with* $\epsilon = \frac{1}{8}$ *and no better ratio can be achieved unless* **P** $=$ **NP**.

References:

- *Approximation Algorithms for NP Hard Problems*, Dorit Hochbaum, ed., PWS, 1997.
- Juraj Hromkovic, *Algorithmics for Hard Problems*, Springer, 2001.
- Sanjeev Arora, "The Approximability of NP-hard Problems", STOC 98, www.cs.princeton.edu/∼arora.