CSE491/596 Lecture Fri. 10/13/23: Complexity, Reductions, and NP

Computational complexity theory is the study of the time, space, and other computational resources needed to solve specified problems, and of the mathematical relationships between problems. The resources are measured on model(s) of computation, of which the deterministic multitape Turing machine (with read-only input tape) is (IMHO) surprisingly realistic. The relationships include reductions. We have already defined running time for nondeterministic as well as deterministic TMs.

Definition: For any time function $t(n)$, using M to mean DTM and N *for NTM*:

1. $\mathsf{DTIME}[t(n)] = \{L(M) : M \text{ runs in time } t(n)\}$
2. $\mathsf{NTIME}[t(n)] = \{L(N) : N \text{ runs in time } t(n)\}$

Convention: For any collection T of time or space bounds, in particular one defined by O-notation, $\mathsf{DTIME}[T]$ means the union of $\mathsf{DTIME}[t(n)]$ over all functions $t(n)$ in T, and so on.

Definition (some of the "Canonical Complexity Classes"):

1. $\mathsf{P} = \mathsf{DTIME}\left[n^{O(1)}\right]$
2. $\mathsf{NP} = \mathsf{NTIME}\left[n^{O(1)}\right]$
3. $\mathsf{E} = \mathsf{DTIME}\left[2^{O(n)}\right]$
4. $\mathsf{EXP} = \mathsf{DTIME}\left[2^{n^{O(1)}}\right]$.

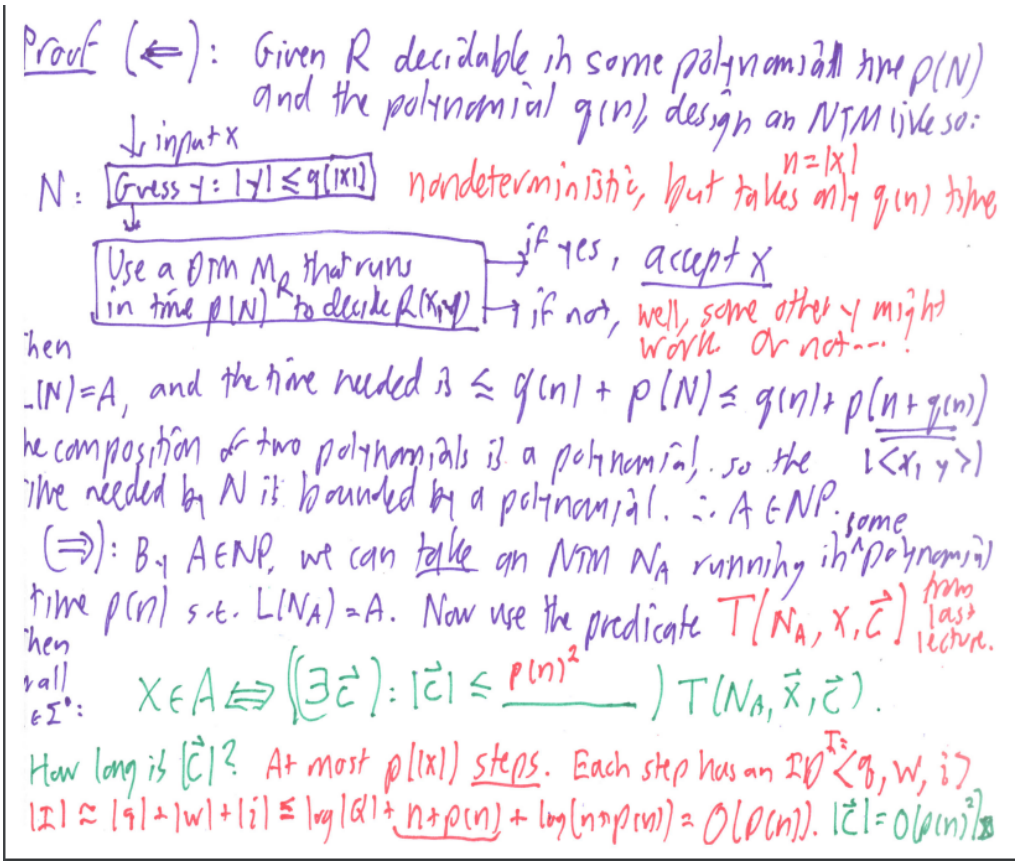
We will cover **space complexity** in detail later. Here is a recap of what was proved about time complexity for the class NP on Wednesday:

Theorem (connecting Theorems 10.2 and 13.12 in Debray's notes): For any language L,

- L is c.e. if and only if there is a decidable predicate $R(x, y)$ such that for all $x \in \Sigma^*$,
$$x \in L \iff \left(\exists y \in \Sigma^*\right) R(x, y).$$
- $L \in \mathsf{NP}$ if and only if there are a polynomial-time decidable predicate $R(x, y)$ and a polynomial $q(n)$ such that for all $x \in \Sigma^*$,
$$x \in L \iff \left(\exists y \in \Sigma^* : |y| \leq q(|x|)\right) R(x, y).$$

In both cases, $R(x, y)$ can be linear-time decidable; indeed, R can be the Kleene T-predicate $T(N, x, \vec{c})$ as defined also for NTMs. (The key difference is the polynomial length bound q, not R.)

In the lecture I wrote it as $VALCOMPS(N, x, \vec{c})$. A sketch recap of the proof:



The second part of the theorem is often used as the *definition* of NP. The polynomial-time DTM V_R is called a **verifier**, and given $x \in L$, any y such that $R(x, y)$ and $|y| \leq q(|x|)$ is called a **witness** (or **certificate**) **for** $x \in L$. It is usually easiest to tell that (the language of) a decision problem belongs to NP by thinking of a witness and its verification. For example:

SAT:
Instance: A logical formula ϕ in variables $x_1, \ldots, x_n$ and operators $\wedge$, $\vee$, $\neg$.
Question: Does there exist a truth assignment $a \in \{0,1\}^n$ such that $\phi(a_1, \ldots, a_n) = 1$?

The assignment cannot have length longer than the formula, and *evaluating* a formula on a given assignment is quick to do. Hunting for a possible *satisfying assignment*, on the other hand, takes up to 2^n tries if there is no better way than brute force.

G3C:
Instance: An undirected graph $G = (V, E)$.
Question: Does there exist a 3-coloring of the nodes of G?

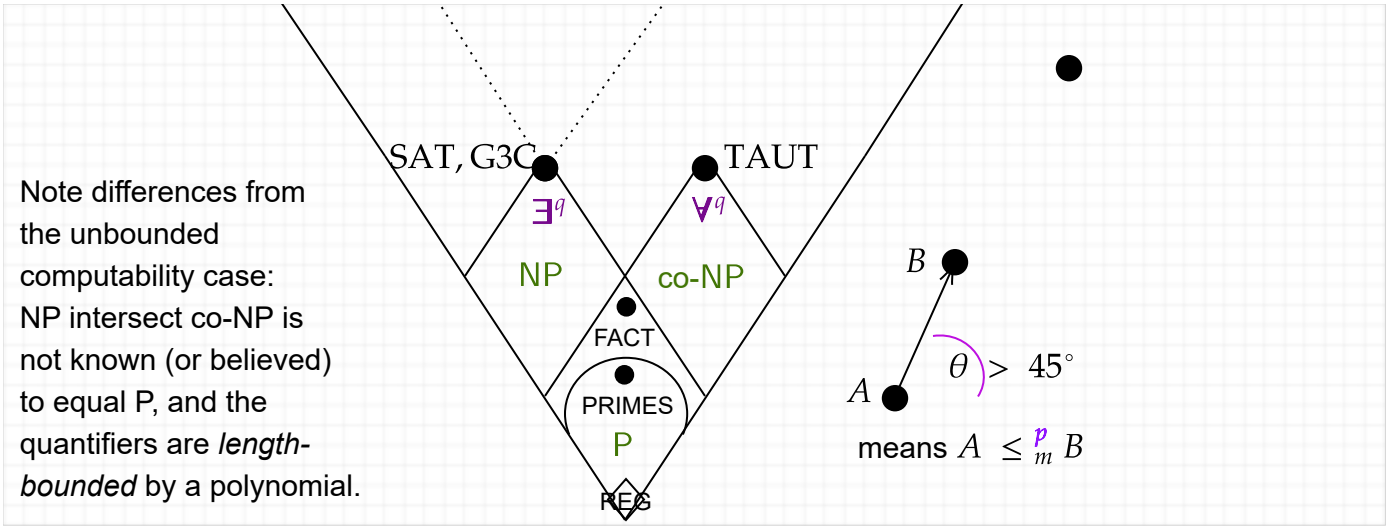
A *3-coloring* is a function $\chi : V \to \{R, G, B\}$ such that for all edges $(u, v) \in E$, $\chi(u) \neq \chi(v)$. The table for χ needs only n entries where $n = |V| \ll N = |G|$, so it has length at most linear in the encoding length N of G (often $N \approx n^2$). And it is easy to *verify* that a *given* coloring χ is correct.

Corollary: For any language L',

- $L' \in$ co - RE if and only if there is a decidable predicate $R'(x, y)$ such that for all $x \in \Sigma^*$,
$$x \in L' \iff (\forall y \in \Sigma^*) R'(x, y).$$
- $L' \in$ co - NP if and only if there are a polynomial-time decidable predicate $R'(x, y)$ and a polynomial $q(n)$ such that for all $x \in \Sigma^*$,
$$x \in L' \iff (\forall y \in \Sigma^* : |y| \leq q(|x|)) R'(x, y).$$

TAUT:
Instance: A Boolean formula ϕ', same as for SAT.
Question: Is ϕ' a **tautology**, that is, true for all assignments?

Note that ϕ is unsatisfiable $\equiv$ every assignment a makes $\phi(a)$ false $\iff$ every assignment a makes $\phi'(a)$ true, where $\phi' = \neg\phi$. Thus TAUT is essentially the complement of SAT.



Let **PRIMES** $= \{2, 3, 5, 7, 11, 13, 17, 19, 23, \ldots\}$ (encoded as, say, $10, 11, 101, 111, 1011, \ldots$)

This language was formally shown to belong to P only in 2004, but had long been known to be "almost there" in numerous senses.

FACT:
Instance: An integer N and an integer k.
Question: Does N have a prime factor p such that $p \leq k$?

If you can always answer yes/no in polynomial time $r(n)$, where $n \approx \log_2 N$ is the number of bits in N, then you can do *binary search* to *find* a factor p of N in time $O(nr(n))$. By doing $N' = n/p$ and repeating you can get the complete factorization of N in polynomial time. This is something that the human race currently does **not** want us to be able to do, as it would (more than Covid?) "destroy the world economy" by shredding the basket in which most of our security eggs are still placed. But to

indicate proximity to this peril, we note:

FACT: **FACT** is in NP $\cap$ co-NP.

Proof: Suppose the answer to an instance $\langle N, k \rangle$ is *yes*. We can verify it by guessing the **unique prime factorization** (u.p.f.) of N as $N = p_1^{a_1} p_2^{a_2} \cdots p_\ell^{a_\ell}$. Although the right-hand side may seem long, ℓ cannot be bigger than the number of bits of N in binary because each p_i is at least 2, and bigger powers only make ℓ have to be smaller. The length of the u.p.f. is $O(n)$. To verify it, one must verify that each p_i is prime---but this is in polynomial time as above---and then simply multiply everything together and check that the result is N. Finally to verify the *yes* answer, check that at least one of the p_i is $\leq k$.

Now suppose the answer to an instance $\langle N, k \rangle$ is *no*. We can verify it by guessing the **unique prime factorization** (u.p.f.) of N as $N = p_1^{a_1} p_2^{a_2} \cdots p_\ell^{a_\ell}$. Although the right-hand side may seem long, ℓ cannot be bigger than the number of bits of N in binary because each p_i is at least 2, and bigger powers only make ℓ have to be smaller. The length of the u.p.f. is $O(n)$. To verify it, one must verify that each p_i is prime---but this is in polynomial time as above---and then simply multiply everything together and check that the result is N. Finally to verify the *no* answer, check that *none* of the p_i is $\leq k$.

Thus we can verify both the *yes* and *no* cases (with the same witness!), so both the language and its complement belong to NP. ☒

This makes the contrast to RE $\cap$ co-RE $=$ REC all the more important. Of course, we don't know NP $\neq$ P either, in contrast to RE $\neq$ REC. What restores much of the analogy is the similarity under reductions and having complete problems. We've seen what comes next already:

Definition: $A \leq_m^p B$ if there is a function $f : \Sigma^* \to \Sigma^*$ that is computable in polynomial time such that for all $x \in \Sigma^*$, $x \in A \iff f(x) \in B$.

This is sometimes called a "Karp reduction" after Richard M. Karp but saying polynomial-time mapping reduction (or many-one reduction) is clear. (There is a corresponding notion called "Cook reduction" after Stephen Cook that uses oracles, but let's *ignore* it for now.)

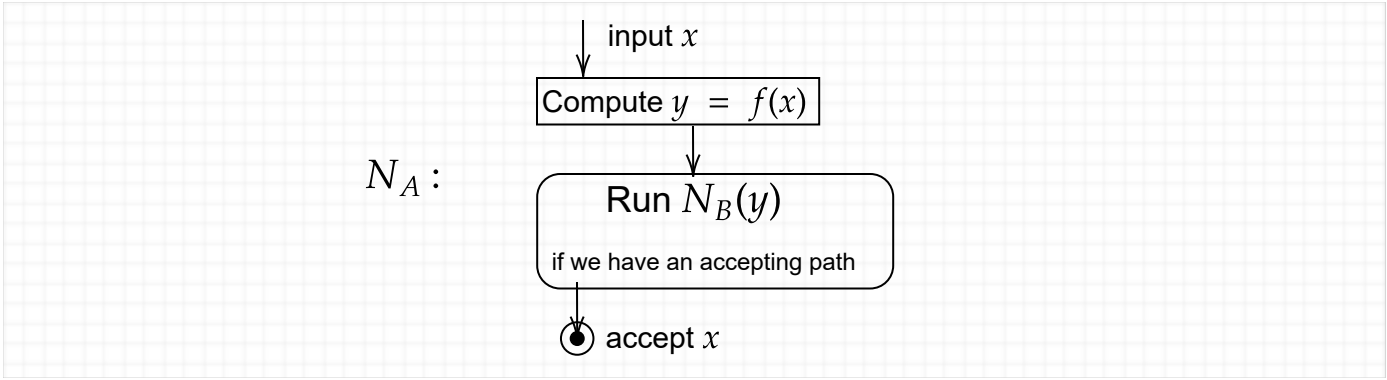
Theorem: Suppose $A \leq_m^p B$. Then:

(a) $B \in$ P $\implies A \in$ P. So $A \notin$ P $\implies B \notin$ P.
(b) $B \in$ NP $\implies A \in$ NP. So $A \notin$ NP $\implies B \notin$ NP.
(c) $B \in$ co-NP $\implies A \in$ co-NP. So $A \notin$ co-NP $\implies B \notin$ co-NP.

The proof is similar to the one with REC and RE and co-RE : We take a machine M_B whose language

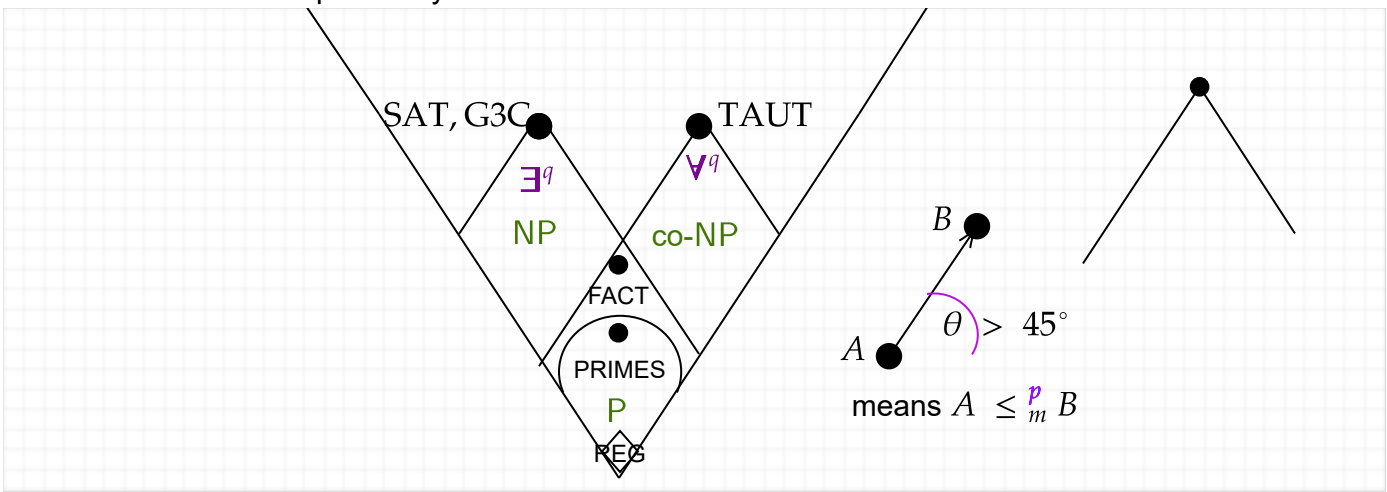
is B and the reduction function f and create the machine M_A that on any input x computes $y = f(x)$ and runs $M_B(y)$, accepting x if and when M_B accepts y. There are two particular details:

- The composition of two polynomials p and q is a polynomial. Thus if f is computable in $p(n)$ tile, then it follows in particular that $|y| \leq p(|x|)$. So if M_B runs in $q(m)$ time, then $M_A(x)$ takes at most $q(p(|x|))$ time, which is a polynomial in $n = |x|$. This shows (a).
- The mapping and timing works in (b) with a polynomial-time NTM N_B in place of a DTM M_B. In that case we get a polynomial-time NTM N_A, which is what we need for $A \in$ NP.



Part (c) again follows simply because $x \in A \iff f(x) \in B$ is the same as $x \notin A \iff f(x) \notin B$. This also means that NP $\cap$ co-NP is likewise **closed downward under** $\leq_m^p$.

This is all summed up visually in the "cone diagram"---except that we don't know if the lines are definite because NP = P is a possibility.



There is one other "grain of salt" that must be taken with all these diagrams: If A and B are two languages in P (technically, other than $\varnothing$ or Σ^* but we sometimes ignore this point), then automatically $A \equiv_m^p B$ (this is a good self-study exercise, including why we have the technicality). Thus to keep up the geometrical intuition of a steep angle meaning $A \leq_m^p B$, we would have to warp the diagram so that P is a single point---squshed even more than how the above shows REG as a tiny subclass of P.