CSE396 Lecture Thu. 5/6: NP-Completeness and the Sweep of Complexity Theory

[Lecture began by redoing the problems below a little more slowly than the end of Tuesday's lecture]

Problems in NP and co-NP

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:

Satisfiability (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. This is apparently hard even when the Boolean formula has a simple form.

Definition. A Boolean formula is in **conjunctive normal form** (CNF) if it is a conjunction of **clauses**

$$\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_m,$$

where each clause C_j is a disjunction of **literals** x_i or $\overline{x}_i$. The formula is in k**-CNF** if each clause has at most k distinct literals (*strictly* so if each has exactly k).

3SAT
Instance: A Boolean formula $\phi(x_1, \ldots, x_n) = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ in 3CNF.
Question: Is there an assignment $\vec{a} = a_1 a_2 \cdots a_n \in \{0,1\}^n$ such that $\phi(a_1, \ldots, a_n) = 1$?

Now for a problem with a different kind of witness:

Graph Three-Coloring (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.

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. But now consider this one:

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

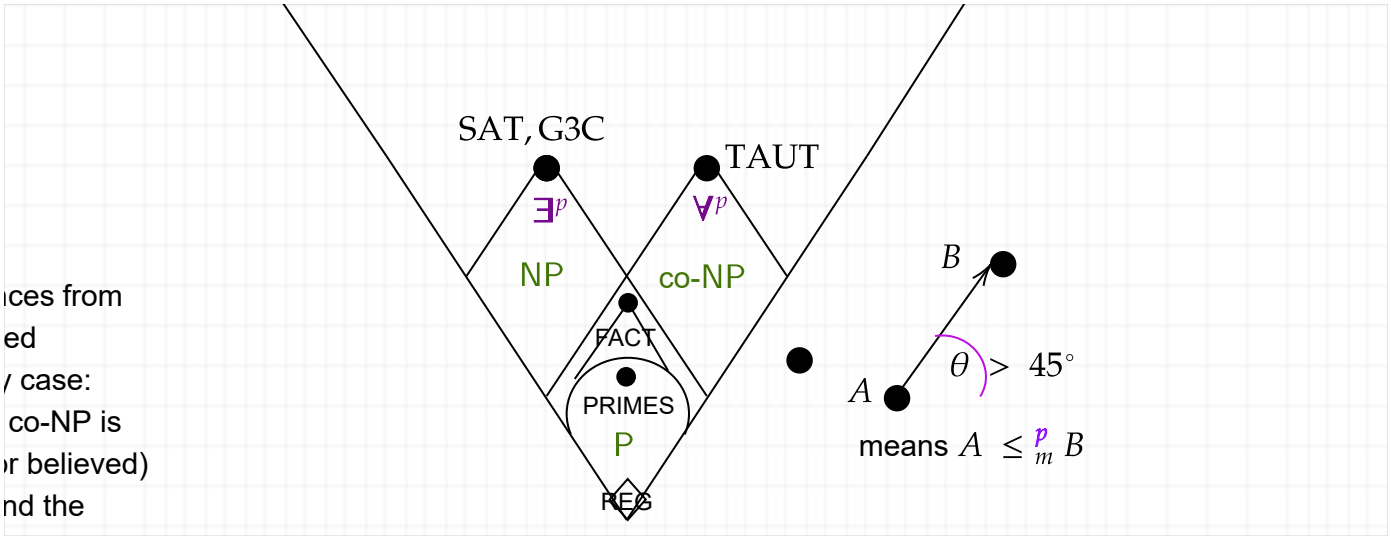
If you can always answer yes/no in polynomial time $r(n)$, where $n \approx \log_2 N$ is the number of bits in N, then you can do *binary search* to *find* a factor p of N in time $O(nr(n))$. By doing $N' = n/p$ and repeating you can get the complete factorization of N in polynomial time. This is something that the human race currently does **not** want us to be able to solve efficiently, as it would (more than Covid?) "destroy the world economy" by shredding the basket in which most of our security eggs are still placed. (This is the gist of the 1992 movie *Sneakers* with Robert Redford heading an all-star cast.) But to indicate proximity to this peril, we note:

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

Proof: The witness for "no" as well as "yes" is the unique prime factorization $N =: p_1^{a1} p_2^{a2} \cdots p_\ell^{a\ell}$. 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 factorization 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 $\leq k$; *no* if none.

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.



ıces from
ed
/ case:
co-NP is
ır believed)
nd the

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 $x \in \Sigma^*$, $x \in A \iff f(x) \in B$.

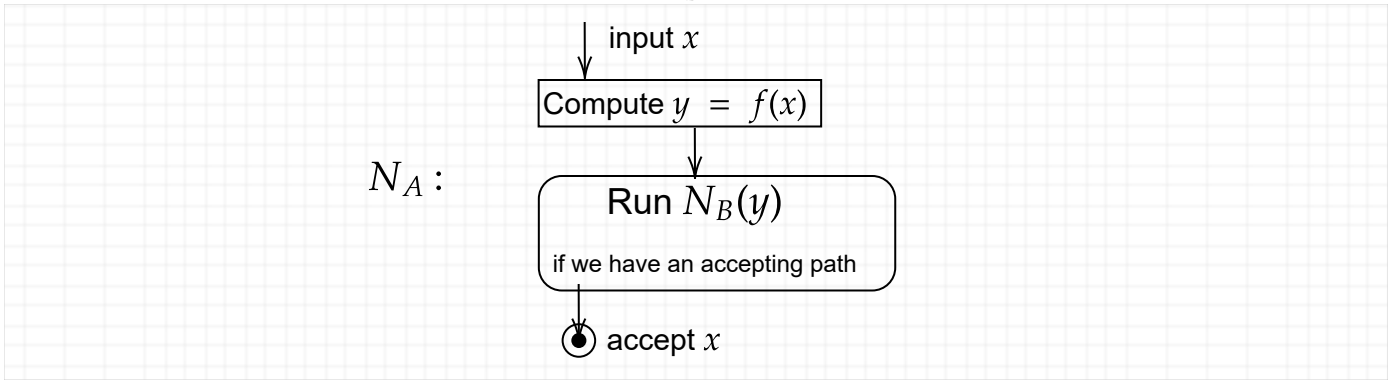
Theorem: Suppose $A \leq_m^p B$. Then:

(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.

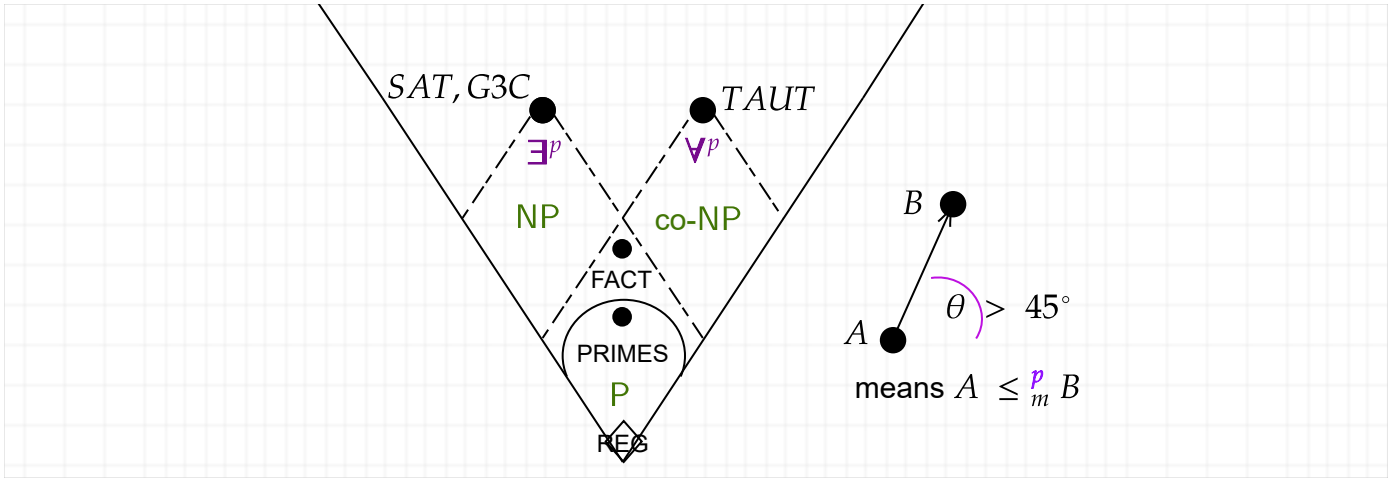
(c) $B \in$ co-NP $\implies A \in$ co-NP. So $A \notin$ co-NP $\implies B \notin$ co-NP.

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. The one extra detail is that *the composition of two polynomials p and q is a polynomial*. Thus if f is computable in $p(n)$ time, then $|y| \leq p(|x|)$. 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 part (a). In part (b) we have an NTM N_B for B:



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.



All the same "visual" logic we had with $\leq_m$ and REC, RE, co-RE works with $\leq_m^p$ for P, NP, and co-NP. Except, what entitles us to put SAT at the top of NP and $TAUT$ atop co-NP? Recall:

Definition: A language B is **complete** for a class C **under** $\leq_m^p$ if $B \in \mathrm{C}$ and for all languages $A \in \mathrm{C}$, $A \leq_m^p B$. When C is the class NP we call B **NP-complete**, with $\leq_m^p$ understood.

The Cook-Levin Theorem

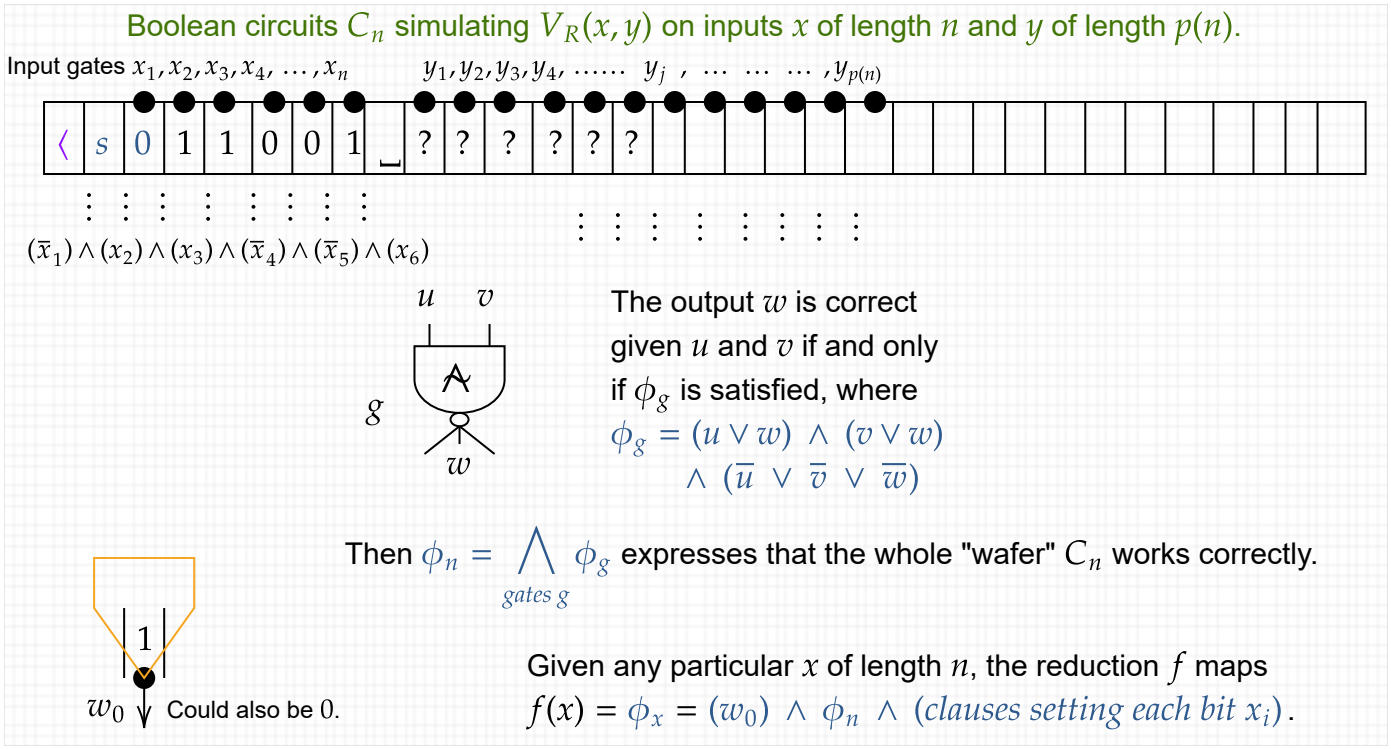
This was proved in 1971 by Steve Cook just up the QEW in Toronto---and he grew up in Buffalo. He just retired from U. Toronto in 2019. We now accept that Leonid Levin, who still teaches at Boston University having come over from Russia in 1978, had a technically stronger version independently, though he did not publish until 1973. Cook gave basically the proof in chapter 7, but we will use the circuit-based proof by Claus-Peter Schnorr in 1978, which is in chapter 9.

Cook-Levin Theorem: **3SAT**, and hence **SAT**, is **NP**-complete.

Proof. We have already seen that **SAT** is in NP and verifying **3SAT** is even easier. Now let any $A \in \mathrm{NP}$ be given. This time we use the "verifier" characterization of NP. We can take a deterministic TM V_R and polynomials p, q such that for all n and x of length n,

$$x \in A \iff (\exists y : |y| = p(n))[V_R \text{ accepts } \langle x, y\rangle]$$

and such that V_R runs in time $q(r)$ where $r = n + p(n)$. Earlier we stated "$|y| \leq p(n)$" as the bound on witnesses, but we can pad them in binary code out to length exactly $p(n)$. Now we "burn the verifier into hardware" as a circuit C_n of size $O\big(q(n + p(n))^2\big)$:



The formula ϕ_x has *witness variables* $y_1, \ldots, y_p$, *wire variables* w_0 plus w_k as needed for any u, v, w of a gate that isn't already a wire for a y_j or x_i, and the input variables $x_1, \ldots, x_n$ whose values get fixed for any particular binary string x. The formula ϕ_x is constructible in essentially $O\big(q(n+p(n))^2\big)$ time by first building C_n given n and the code of the verifier, translating every NAND gate of C_n into clauses as above in one simple pass, and finally *and*-ing (w_0) and the clauses (x_i) if bit i of x is 1, $(\overline{x}_i)$ if the bit is 0. Then

$$\phi_x \in 3SAT \iff (\exists y, \vec{w})\phi_x(x, y, \vec{w}) = 1 \iff (\exists y)C_n(x, y) = 1 \iff (\exists y)V_R(x, y) = 1 \iff x \in A\,.$$

So $f(x) = \phi_x$ reduces A to 3SAT. Since $A \in$ NP is arbitrary, and $3SAT \in$ NP, it is NP-complete. ☒

The meaning of the Cook-Levin theorem is that "logic is universal." The further amazing fact is that myriad other problems---in all walks of nature---embody the universality of logic in ways that make them also NP-complete. Here are two more examples:

CLIQUE

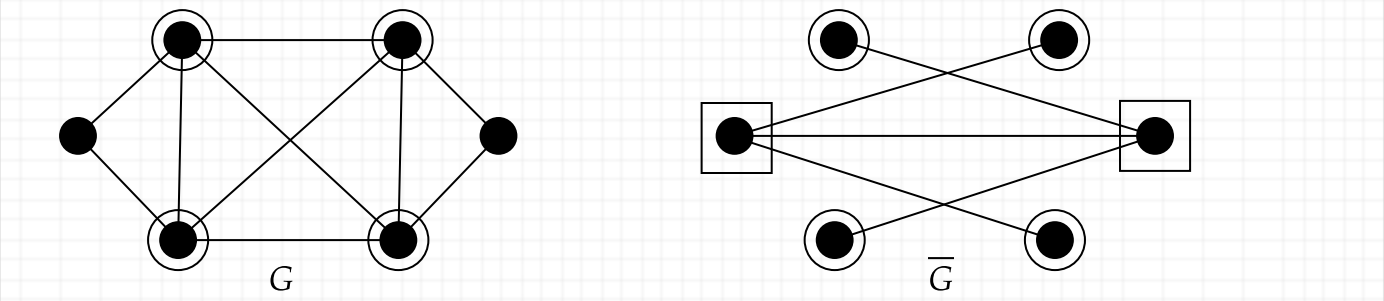
Instance: An undirected graph $G = (V, E)$ and a number $k \geq 1$.
Question: Does there exist a set $S \subseteq V$ of k (or more) nodes such that for each pair $u, v \in S$, (u, v) is an edge in E?

INDEPENDENT SET

Instance: An undirected graph $G = (V, E)$ and a number $k \geq 1$.
Question: Does there exist a set $S \subseteq V$ of k (or more) nodes such that for each pair $u, v \in S$, (u, v) is **not** an edge in E?

The languages of these problems are *not* complements of each other, despite their differing by just the word "not" at the end. Both languages are in NP with S as the witness. They are not believed to be in P because with $n = |V|$, there are 2^n subsets S that may need to be considered. A polynomial-time algorithm cannot try each one. Any given S, however, can be verified by looking up at most n^2 possible edges (u, v). So the body is a polynomial-time decidable predicate $R(G, S)$. What gets complemented is not even this predicate but *the graph* G, as expressed by this fact:

G has a clique of size $k \iff$ the complementary graph $\overline{G}$ has an independent set of size k.



Therefore, the simple reduction function $f(\langle G, k\rangle) = (\langle \overline{G}, k\rangle)$ reduces CLIQUE to IND SET and also vice-versa, so the problems are $\equiv_m^p$ equivalent. A second fact yields a second equivalence:

The complement of an independent set S in G is a set S' of nodes such that every edge involves a node in S'. Such an S' is called (somewhat midleadingly, IMHO) a **vertex cover**. Therefore:

G has an independent set of size (at least) $k \iff$ G has a vertex cover of size (at most) $n - k$.

Note that the graph G stays the same; instead we flip around the target number from k nodes to $|V| - k$ nodes. In practice, when we're trying to optimize, we want to *maximize* cliques and independent sets and *minimize* vertex covers. The latter gives rise to this decision problem:

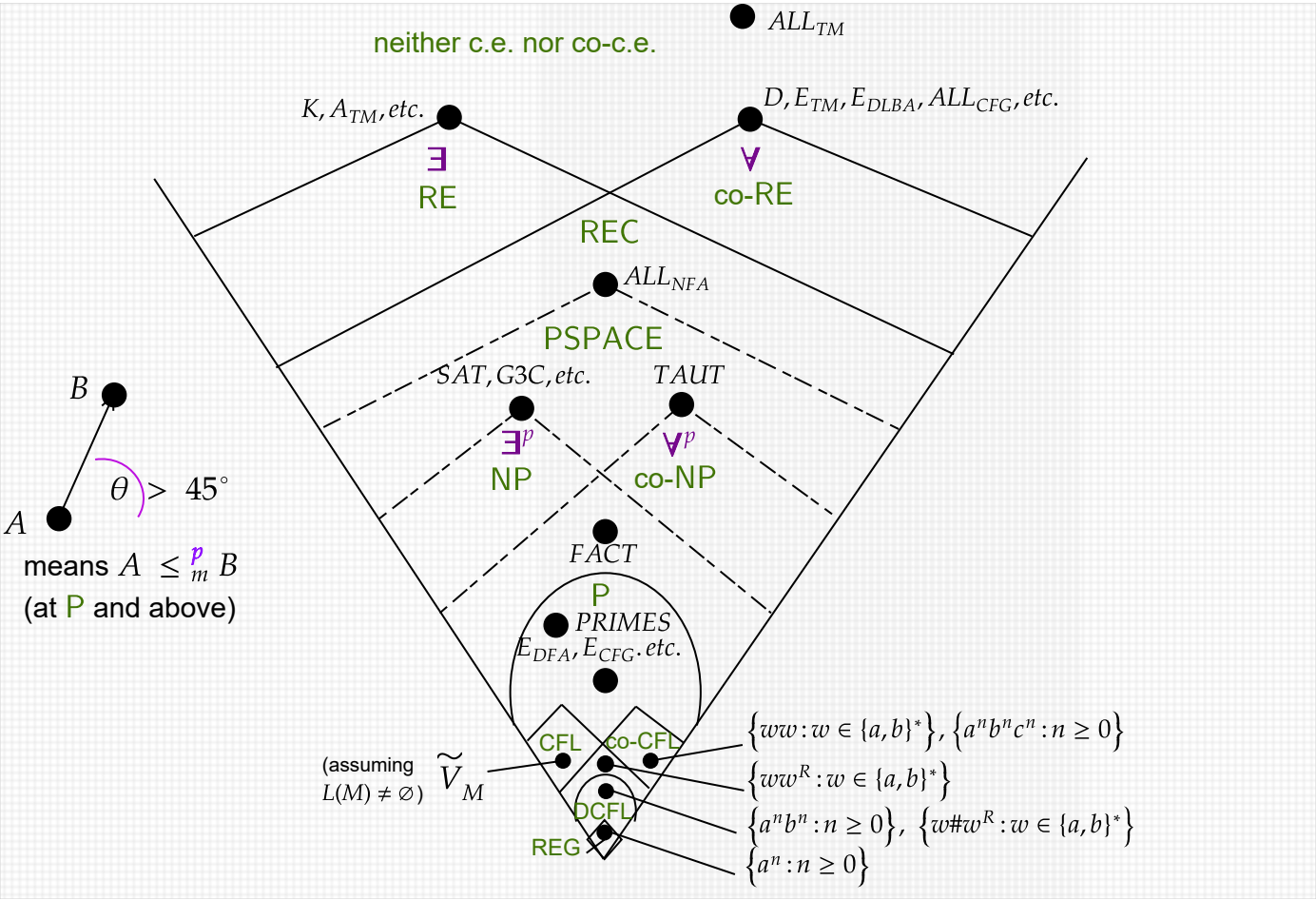
VERTEX COVER (VC)
Instance: A graph G and a number $\ell \geq 1$.
Question: Does G have a vertex cover of size (at most) ℓ?

Then IND SET and VC reduce to each other via the reduction $g(G, k) = (G, n - k)$ (where it is understood that $G = (V, E)$ and $n = |V|$.)

Theorem: $3SAT \leq_m^p IND\ SET$, so all three graph problems are NP-complete. ⊟

The box-minus means we'll skip the proof from chapter 7, though it is done in CSE491/596 and also sometimes in CSE331. Also **Graph 3-Coloring** is NP-complete, as are thousands of other problems in NP: the Traveling Salesperson Problem (TSP), Hamilton Circuit (HAM), and more.

Here is a flyover of the whole range of computational problems we have covered in the course:

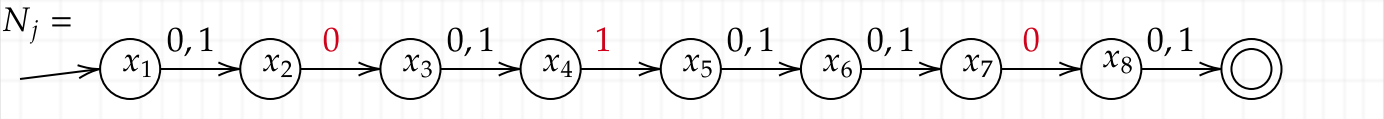


As and if time allows, discuss some of the following:

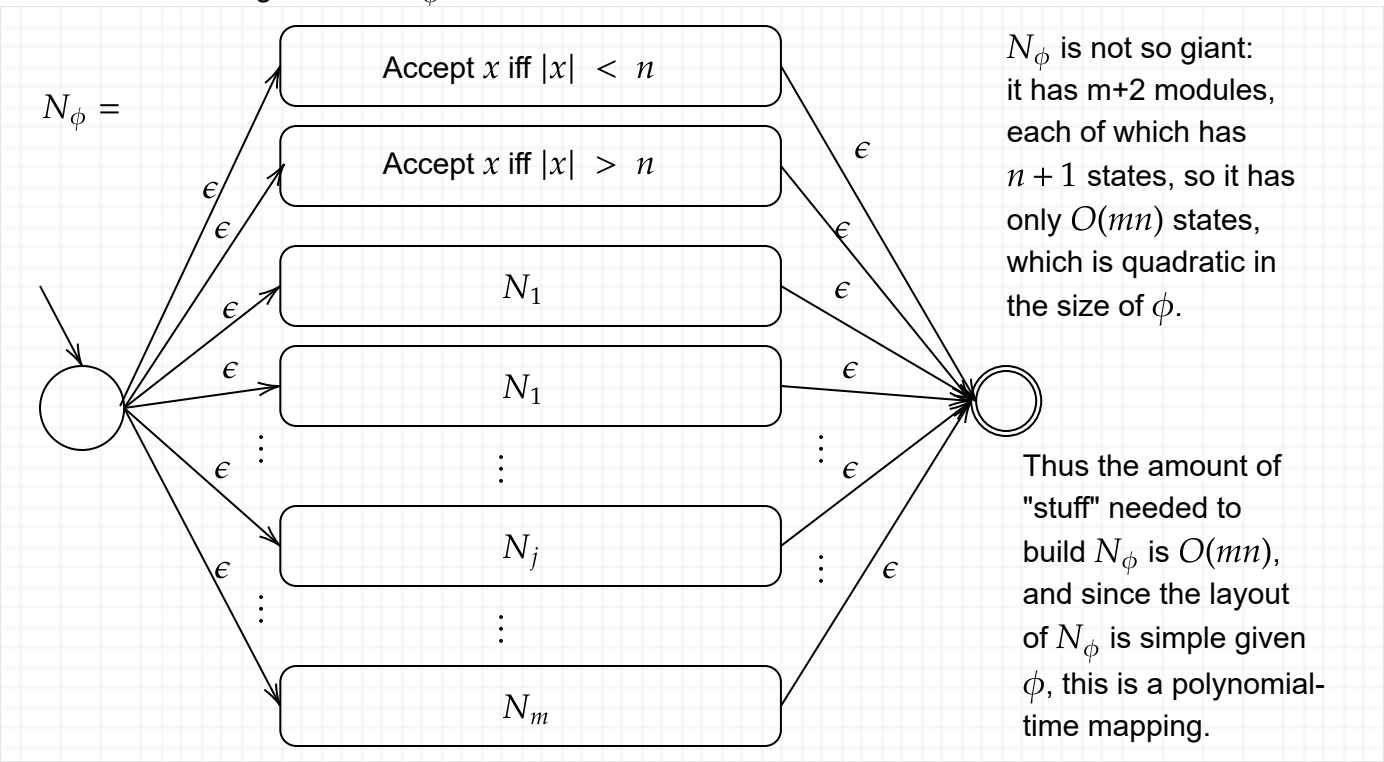
- The fact that ALL_{NFA} is **NP-hard** but not even known to belong to NP or co-NP---indeed, it is complete for PSPACE under $\leq_m^p$. The problems ALL_{Regexp}, EQ_{NFA}, and EQ_{Regexp} are likewise **PSPACE-complete**. (They also belong to DLBA, which along with NLBA is a subclass of PSPACE, so they are complete for those subclasses as well.) The upshot is that whether programs are correct---or even whether relatively simple kinds of machines or expressions are comprehensive---can be **hard** to tell, *especially if you don't provide any comments that help to prove it is correct*. One might expect ALL_{NFA} to belong to co-NP because of "all". Let's look at the complementary problem $NOTALL_{NFA}$ and its language

$$NOTALL_{NFA} = \{\langle N \rangle : (\exists x)[N \text{ does not accept } x]\}.$$

  This has "exists"; why doesn't it belong to NP? The reason is that we don't necessarily have the polynomial length bound on $|x|$ in terms of $|\langle N \rangle|$. There are cases of NFAs that fail to accept some strings way longer than their number of states. But we can show that $NOTALL_{NFA}$ is NP-hard: For a 3CNF clause C_j such as $(x_2 \vee \overline{x}_4 \vee x_7)$ (say with $n = 8$ variables in the given formula $\phi(x_1, \ldots, x_n) = C_1 \wedge C_2 \wedge \cdots \wedge C_m$), we can build an NFA N_j to accept the assignments $a \in \{0,1\}^n$ that do *not* satisfy C_j. In our example:



Now build a giant NFA N_ϕ like so:



N_ϕ is not so giant: it has m+2 modules, each of which has $n+1$ states, so it has only $O(mn)$ states, which is quadratic in the size of ϕ.

Thus the amount of "stuff" needed to build N_ϕ is $O(mn)$, and since the layout of N_ϕ is simple given ϕ, this is a polynomial-time mapping.

Then a satifying assignment a of ϕ is a string of length n that N_ϕ does **not** accept, so $\langle N_\phi \rangle \in NOTALL_{NFA} \iff \phi \in 3SAT$.

- The frontier of program-analysis problems becoming undecidable is at CFGs and (D)PDAs. For another example, consider traces of the form $\vec{c'} = \langle I_0(x), I_1^R, I_2, I_3^R, \ldots, I_{t-2}, I_{t-1}^R, I_t \rangle$ (if t is even) in which every odd-numbered ID is written in reverse. For any TM M, define V'_M to be the set of valid accepting traces in this form. A DPDA P_M can push $I_0(x)$ and compare it against I_1^R in palindrome fashion while popping. But then its stack is empty, so it can't check $I_1 \vdash_M I_2$ on the fly the same way. But it can handle I_2 and I_3^R. Meanwhile, a second DPDA P'_M can skip over I_0 and match I_1^R to I_2, I_3^R to I_4, and so on. Then $V'_M = L(P_M) \cap L(P'_M)$, and so $L(M) = \varnothing \iff V'_M = \varnothing \iff L(P_M) \cap L(P'_M) = \varnothing$. In consequence, the "$E \cap_{DCFL}$" problem of whether the intersection of two DCFLs is empty is undecidable, ditto "$E \cap_{CFL}$" for two CFLs. Thus given two CFGs G_1, G_2 it is decidable whether $L(G_1) = \varnothing$ or $L(G_2) = \varnothing$ individually, but not whether $L(G_1) \cap L(G_2) = \varnothing$.

- We don't even know whether PSPACE is different from P, let alone whether NP ≠ P. We know that EXP properly contains P. EXP also contains PSPACE, and hence contains both NP and co-NP, but we don't even know whether EXP is different from NP ∩ co-NP. At least one of the statements EXP ≠ PSPACE and PSPACE ≠ P must be true, but we don't know which.

- The factoring problem $FACT$ is believed not to be NP-complete, because if it is, then NP = co-NP. But it is believed to be *hard* in a practical time sense, indeed to require time $2^{d^{\Omega(1)}}$ for most products $N = pq$ of two d-digit primes p and q. Those primes are secret keys of the **Rivest-Shamir-Adleman** (**RSA**) cryptosystem, which underlies **PGP**, and many other forms of cryptography including most implementations of **blockchain** rely on $FACT$ being *hard*. But it does not have the bedrock of NP-completeness to back up its security.

- The **Church-Turing Thesis** (CTT) extends to assert that no physical device will ever be theoretically capable to accept languages outside RE, nor to decide problems outside REC. I was a fellow "fellow" of David Deutsch at Oxford in 1985 when he challenged this by claiming that a **quantum computer** could decide the Halting Problem. His claim was refuted, but a recent theorem that two "quantum provers" can "kind-of" decide HP_{TM} indicates that the claim wasn't wacky. And he's having at least the next-to-last laugh because...
- The "**Polynomial-Time CTT**" says that all problems we will ever be practically able to solve in all cases, with whatever hardware, belong to P in their language versions. This was challenged in 1993-94 by Peter Shor's theorem that a quantum computer can factor d-digit numbers in $O(d^2)$ quantum effort with virtual certainty. We've not yet come close to building a **scalable universal** quantum computer, but the theory is clear. This challenges not only national security and the polynomial-time CTT, but also philosophical issues about whether the *logos* of nature is inherently *lexical*.