CSE396 S26 Last Lecture, Tue. Week 15: The Grand Sweep of NP-Completeness And More

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. We will begin with (and emphasize) problems that are "SAT-like" in that the embodiment of logic is on the surface and transparent. Then we will skim through examples in section 7.5 involving graphs and sets of numbers where the embedding of logic is more subtle.

Example: Solving Equations

Solving Equations

Instance: A set S of polynomial equations $p_1(x_1, \ldots, x_n) = 0$, ..., $p_s(x_1, \ldots, x_n) = 0$ where each p_k is a polynomial in variables $x_1, \ldots, x_n$ with integer coefficients.
Question: Do the equations have a common solution?

To prove this NP-hard, we begin by observing that we make the set S include the equations $x_1^2 - x_1 = 0$ through $x_n^2 - x_n = 0$, then each x_i is forced to be 0 or 1. The possible solutions are then correspond to possible Boolean truth assignments to a formula. Now the rest of the idea comes into view: we can make every clause C_j into an equivalent equation p_j, so that C_j is satisfied iff the equation is made true.

For the details, suppose the clause C_j is $(x_1 \vee x_2 \vee x_3)$. Then we actually want to make $p_j(0, 0, 0)$ not be 0, while we want all other cases to be 0. The trick is to make

$$p_j(x_1, x_2, x_3) \;=\; (1 - x_1)(1 - x_2)(1 - x_3).$$

If, on the other hand, the clause C_k is, say, $(\overline{x}_2 \vee x_4 \vee \overline{x}_5)$, then the equation becomes

$$p_k(x_2, x_4, x_5) \;=\; x_2(1 - x_4)x_5.$$

The reduction function $g(\phi) = S$ where $\phi = C_1 \wedge \cdots \wedge C_m$ is now clear: it translates each C_j into p_j and appends the extra equations $x_1^2 - x_1 = 0$ through $x_n^2 - x_n = 0$, making $s = m + n$ equations in all. Thus the **Solving Equations** problem is NP-hard, by reduction from 3SAT. (That it belongs to NP, and so is NP-complete, is cold comfort. Solving a bunch of equations is hard!)

Before moving on, we note some technical facts about cases of 3SAT and the formulas in the Cook-Levin proof. First, the clauses in the proof either had all-positive literals or all-negated literals. The equation corresponding to the latter kind of clause, such as $(\overline{u} \vee \overline{v} \vee \overline{w})$, is just the monomial equation $uvw = 0$. You could call the product $(1 - x_1)(1 - x_2)(1 - x_3)$ obtained for the all-positive clause $(x_1 \vee x_2 \vee x_3)$ above an "anti-monomial." Thus equation solving remains NP-hard even when all the equations are monomials or anti-monomials ... except for the extra equations, which factor as $x_1(1 - x_1) = 0$ up through $x_n(1 - x_n) = 0$.

Second, I defined **3CNF** as meaning exactly three (different) literals per clause, but the Cook-Levin proof used up-to-three. Consider, however, the following 3CNF formula with 5 variables and 20 clauses (this is not in the text):

$$\Phi = (z_1 \vee z_2 \vee z_3) \wedge (z_1 \vee z_2 \vee z_4) \wedge (z_1 \vee z_2 \vee z_5) \wedge \cdots \wedge (z_3 \vee z_4 \vee z_5)$$
$$\wedge (\overline{z}_1 \vee \overline{z}_2 \vee \overline{z}_3) \wedge (\overline{z}_1 \vee \overline{z}_2 \vee \overline{z}_4) \wedge (\overline{z}_1 \vee \overline{z}_2 \vee \overline{z}_5) \wedge \cdots \wedge (\overline{z}_3 \vee \overline{z}_4 \vee \overline{z}_5).$$

That is, Φ has all combos of three variables, once all-positive and once all-negative. Two facts:

- Φ is **unsatisfiable**: Every truth assignment $(a_1, a_2, a_3, a_4, a_5)$ either makes three variables false or makes three variables true. In the former case, if say $a_2 = a_4 = a_5 = 0$, then the positive clause $(z_2 \vee z_4 \vee z_5)$ isn't satisfied. In the latter case, the corresponding negative clause is unsatisfied.
- If you make Φ' by deleting the first clause $(z_1 \vee z_2 \vee z_3)$, then Φ' is **uniquely satisfied** by the assignment $a_1 = a_2 = a_3 = 0, a_4 = a_5 = 1$.

The upshot is that if you conjoin Φ' onto the formula in the Cook-Levin proof then you can pad, say, $(u \vee w) \wedge (v \vee w) \wedge (w_o) \wedge (\overline{x}_1)$ out to be

$$(u \vee w \vee z_1) \wedge (v \vee w \vee z_1) \wedge (w_o \vee z_1 \vee z_2) \wedge (\overline{x}_1 \vee \overline{z}_4 \vee \overline{z}_5).$$

The way Φ' forces the values of the z-variables makes no change in the overall logic. Henceforth, we can presume that a Boolean formula ϕ that we use as the *source* of our reductions to other problems is in strict 3CNF form with all-positive or all-negative clauses. That is, we can take the following as our neatest NP-complete problem to use as the **source** of reductions:

POS-NEG 3SAT

Instance: A strict 3CNF formula $\phi(x_1, \ldots, x_n) = C_1 \wedge \cdots \wedge C_m$ in which each clause C_j has three positive literals or has three negative literals.
Question: Does ϕ have a satisfying assignment?

Example: Independent Sets and Cliques

Here are two graph problems:

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?

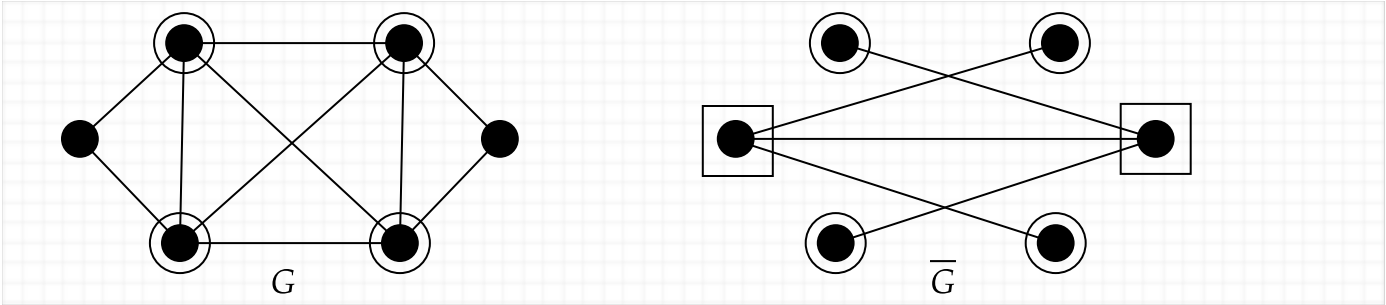
INDEPENDENT SET

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.

Our textbook reduces **3SAT** $\leq_m^p$ **CLIQUE**, even before giving the Cook-Levin proof in chapter 7. We will reduce **3SAT** $\leq_m^p$ **IND SET** instead. The conclusion about **CLIQUE** then follows from **IND SET** $\leq_m^p$ **CLIQUE** as above and reductions being transitive.

Theorem: **3SAT** $\leq_m^p$ **IND SET**, and since **IND SET** is in NP, **IND SET** is NP-complete.

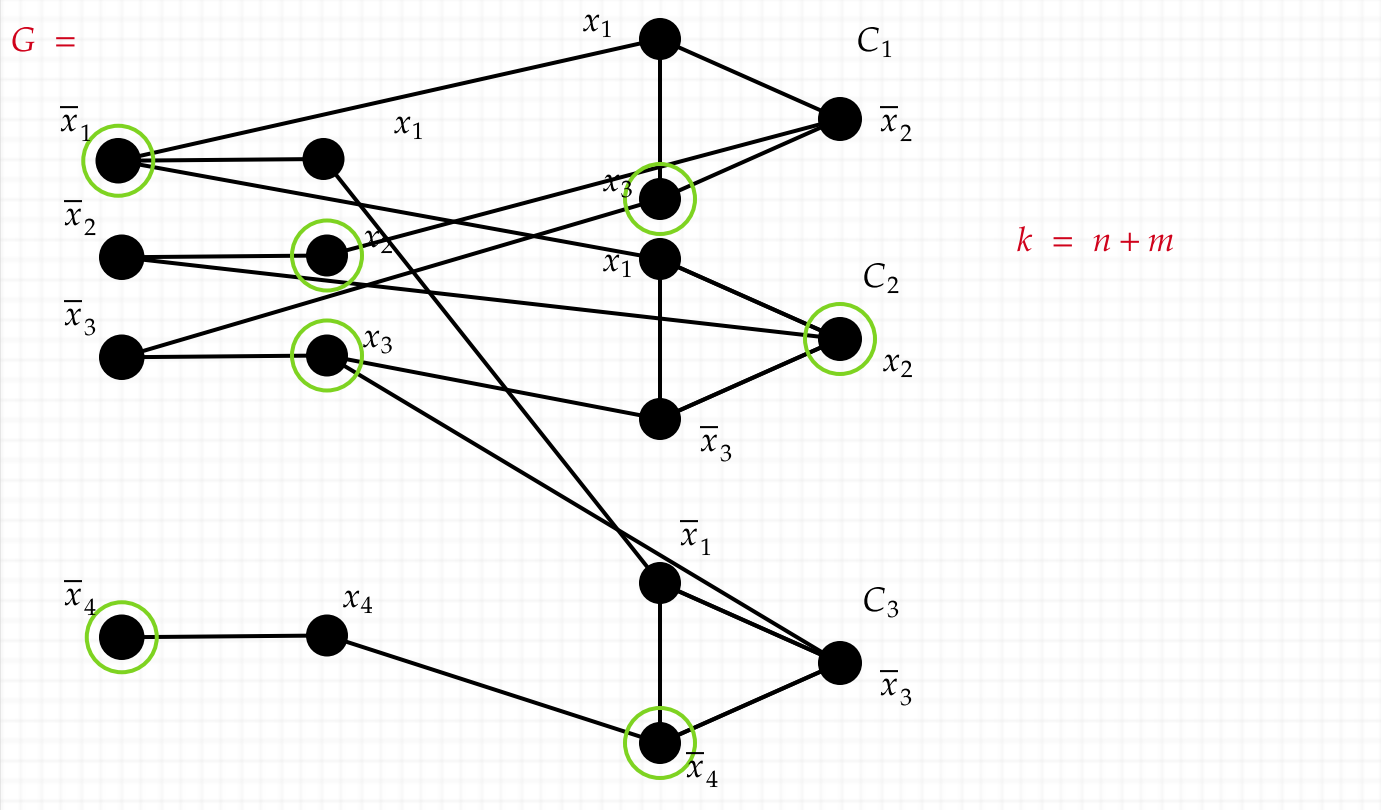
Proof (sketch): Given any 3CNF formula $\phi(x_1, \ldots, x_n) = C_1 \wedge C_2 \wedge \cdots \wedge C_m$, we build a graph $G = (V, E)$ and set $k = m + n$ to get an equivalent instance (G, k) of **Independent Set** as follows:

- V consists of $2n$ "rung nodes" labeled $x_1, \overline{x}_1, x_2, \overline{x}_2, \ldots, x_n, \overline{x}_n$ and $3m$ "clause triangle nodes". (It is exactly $3m$ nodes if ϕ is in "strict 3CNF", which you are allowed to assume.)
- E first has n "rung edges", each between some x_i and its negation $\overline{x}_i$.
- Then E has $3m$ "clause gadget edges" to make a triangle for each clause.
- Finally and most critically, E has $3m$ "crossing edges". For each occurrence of a positive literal x_i in a clause gadget, the edge goes to the negated $\overline{x}_i$ in its "rung". (The edges and whole

graph are undirected.) For each occurrence of a negative literal $\overline{x}_i$ in a clause gadget, the edge goes to the positive x_i in its "rung".

This finishes the **construction** of $G = (V, E)$ and k in general. (We did not need the "POS-NEG" feature.) Here is an example and picture:

$$\phi = (x_1 \vee \overline{x}_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \overline{x}_3) \wedge (\overline{x}_1 \vee \overline{x}_3 \vee \overline{x}_4)$$



Note that a choice of vertices for S is not part of G---not part of the reduction function f itself. It is only part of the analysis of why the reduction is correct.

Complexity: G can be built with simple passes over ϕ. [Usually this can be done in a sentence or two.]

Correctness: We want to show both of these implications:

- **If** ϕ has a satisfying assignment $a = (a_1, a_2, \ldots, a_n)$, **then** from a we can make choices of the existentially questioned object (in this case, an independent set) to meet the stated requirements (here, including meeting the size target k).
- **If** there is an object (i.e., "witness") that answers "yes" to the problem, **then** from that object we can find a satisfying assignment to ϕ.

Together, these show that ϕ is satisfiable $\iff$ the answer to the target instance (G, k) is "yes". This completes the requirements of reducing 3SAT to the target problem (by a polynomial-time many-one reduction), so the target problem is NP-hard. Since it belongs to NP, it is NP-complete. ☒

To illustrate the analysis, note that the example formula ϕ is satisfiable---in fact, it has many satisfying assignments. The diagram shows $a = 0110$. If we used the big formula Φ above with $n = 5$ and $m = 20$, we'd get a graph of $10 + 60 = 70$ nodes and $5 + 60 + 60 = 125$ edges that does *not* have an independent set of size $k = n + m = 25$, because Φ is unsatisfiable.

More Graph Problems

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?

Theorem: VC $\equiv_m^p$ IND SET $\equiv_m^p$ CLIQUE, so all three problems are NP-complete.

Proof: IND SET and VC reduce to each other via the same reduction $g(G, k) = (G, n - k)$ (where it is understood that $G = (V, E)$ and $n = |V|$). The rest we've seen before. ☒

We mentioned the **Graph 3-Coloring** problem last week. It, too, is NP-complete. Other problems treated in the text are the Hamilton Circuit problem (HAM), Traveling Salesperson Problem (TSP), the **Subset Sum** problem, and many, many more. We will, however, step out of **NP** to cover one of the last remaining problems that are *about* automata covered earlier in this course.

The ALL_{NFA} Problem

Given an NFA $N = (Q, \Sigma, \delta, s, F)$, is $L(N) = \Sigma^*$? Let $n = |Q|$ be the number of states. We may presume Σ is binary. If the machine were a DFA, then it would have $2n$ instructions and the whole size of the encoding $\langle N \rangle$ would be $O(n)$---well technically $O(n \log n)$ because the vertex labels are binary numbers. An NFA, however, can have many more instructions. Nevertheless, in the most natural instances of NFAs the nondeterminism is sparing, with only binary branching and maybe not at all states. So we can use "n" as the complexity size parameter of the NFA as well.

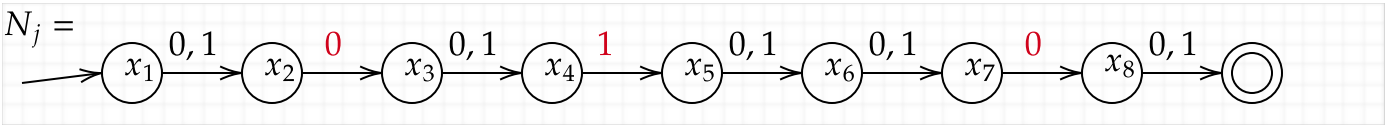
One might expect ALL_{NFA} to belong to co-NP because of "all". Let's look at the complementary problem $NOTALL_{NFA}$ and its language

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

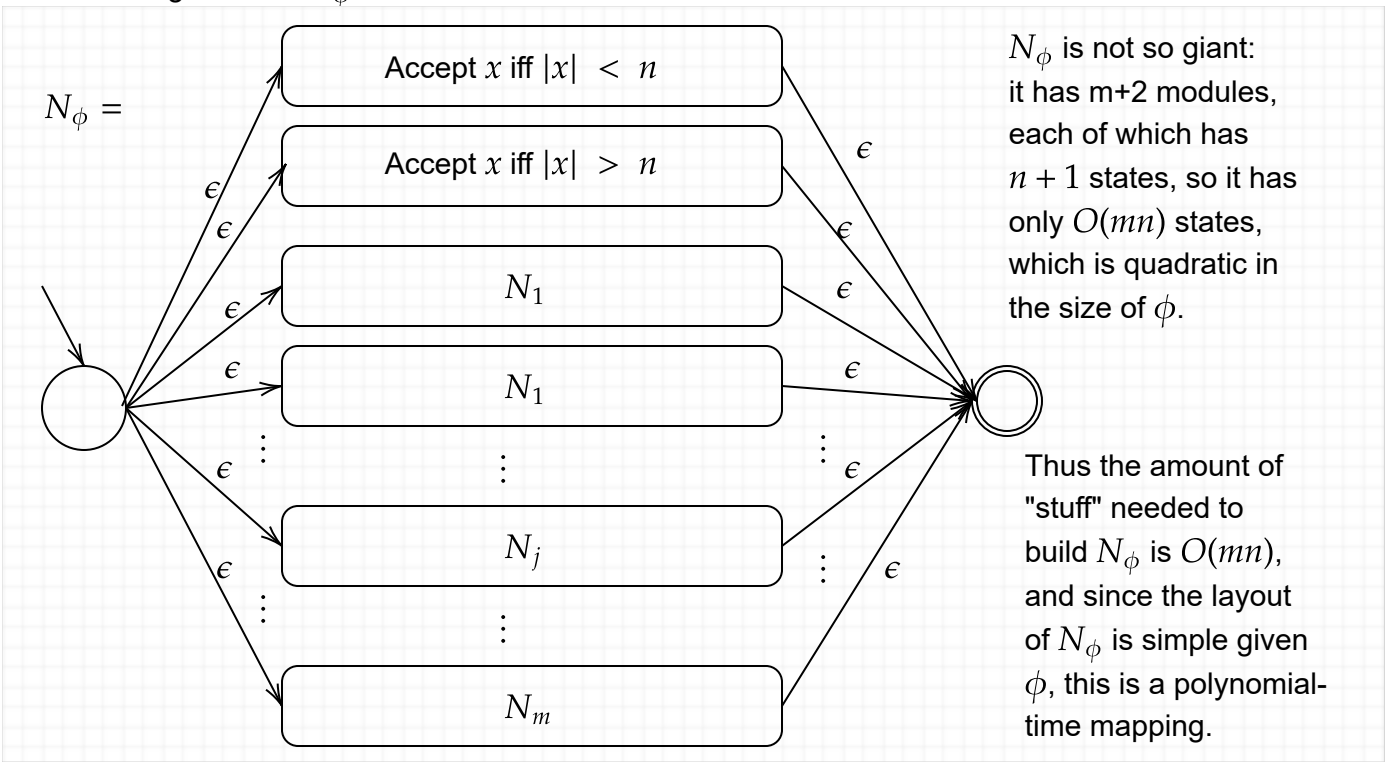
This has "exists"; why doesn't it belong to NP? The reason is that we don't necessarily have the polynomial length bound on $|x|$ in terms of $|\langle N \rangle|$. There are cases of NFAs that fail to accept some strings way longer than their number of states. For example, let $p_1, p_2, \ldots, p_k$ be k different prime numbers. We can build an NFA N with $1 + p_1 + \cdots + p_k$ states such that the shortest string it does *not* accept has length $p_1 p_2 \cdots p_k$. Example for $2, 3, 5$. But we can show:

Theorem: **3SAT** $\leq_m^p NOTALL_{NFA}$, so $NOTALL_{NFA}$ is **NP**-hard and ALL_{NFA} is **co-NP**-hard.

Proof: Given a 3CNF formula $\phi(x_1, \ldots, x_n) = C_1 \wedge C_2 \wedge \cdots \wedge C_m$, we take each individual clause C_j and first build an NFA N_j to accept the assignments $a \in \{0,1\}^n$ that do *not* satisfy C_j. For instance, if C_j is $(x_2 \vee \overline{x}_4 \vee x_7)$ and there are $n = 8$ overall, then



Now build a giant NFA N_ϕ like so:



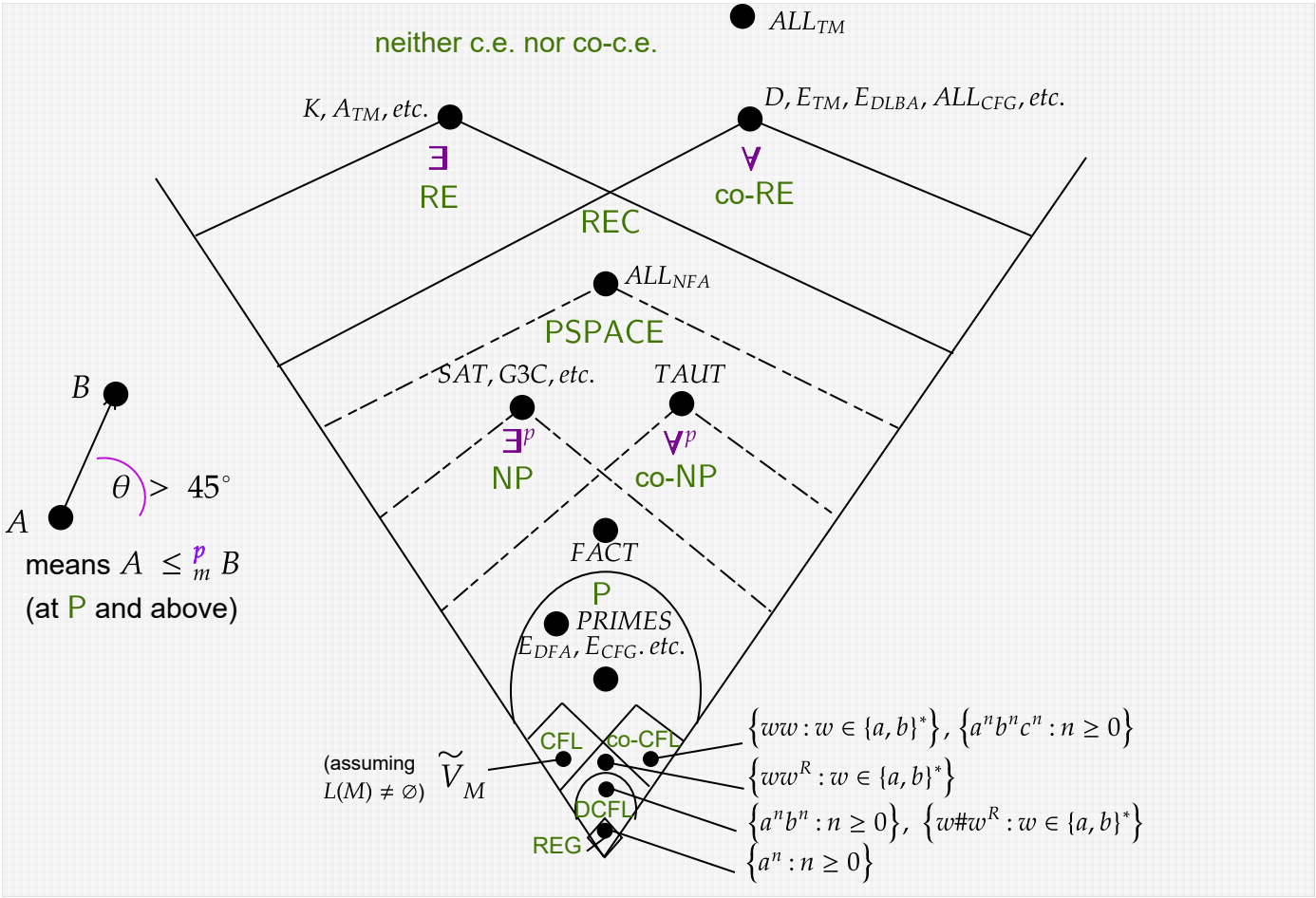
N_ϕ is not so giant: it has m+2 modules, each of which has $n+1$ states, so it has only $O(mn)$ states, which is quadratic in the size of ϕ.

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 $\langle N_\phi \rangle \in NOTALL_{NFA} \iff \phi \in 3SAT$. ⊠

In fact, **both** ALL_{NFA} and $NOTALL_{NFA}$ are complete for the class **PSPACE** of languages accepted by deterministic Turing machines that run in polynomial space. Those include all deterministic LBAs. In fact, nondeterministic machines that run in space $s(n)$ can be simulated deterministically in space $O\big(s(n)^2\big)$, so before you can even define a class "**NPSPACE**" it turns out to be equal to **PSPACE**. Thus **PSPACE** includes all NLBA languages too.

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



Grand Summary

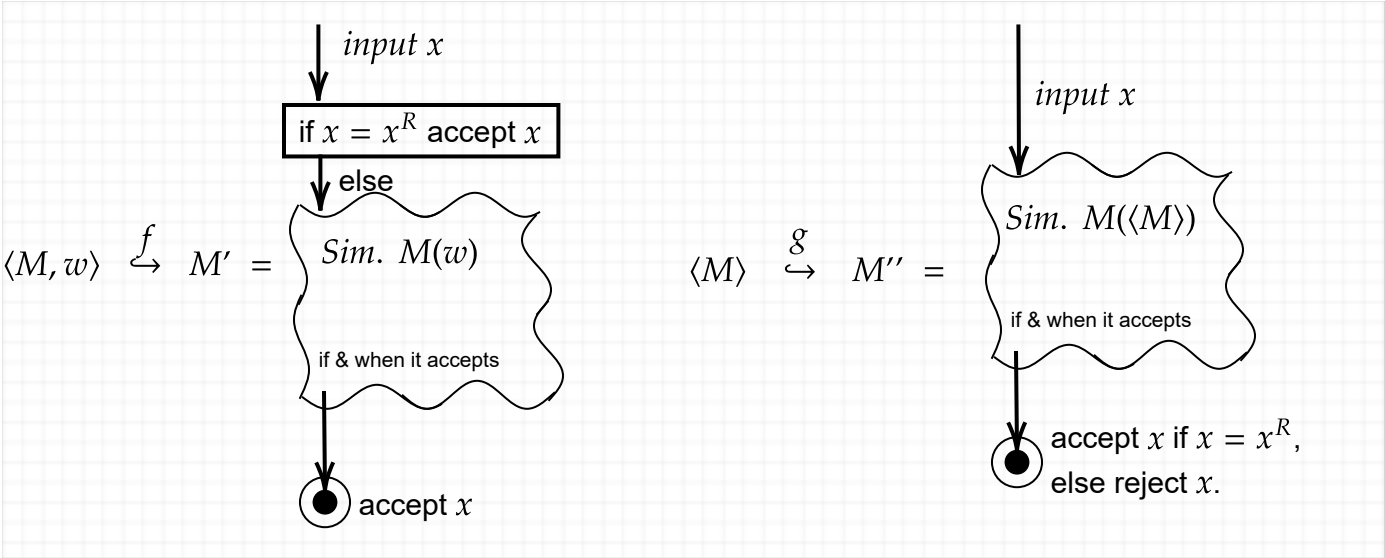
What are we to make of the twin pillars of undecidability and NP-hardness? Along with the fact of our knowing **RE** ≠ **REC** but currently not knowing whether **NP** ≠ **P**, the principles associated to the former are much more clear-cut. Here are some:

1. *Every optional or erroneous behavior of programs is undecidable.* This is the upshot of the "Useful Class" and "Hang" problems in lecture and HW. For contrast, the behavior in an

assembly program P with numbered lines of control jumping to an earlier line is not optional. It is necessary to implement any loop. And it is decidable: just run P and it will either go straight thru to its last line and halt or will jump back at some point.

2. *Every nontrivial "extensional" property of programs*---meaning it depends only on the language L it accepts or the (partial) function f it computes---*is undecidable.* The only exception is the trivial property, "given $\langle M \rangle$, is $L(M)$ c.e.?" which is always answered yes.

For example, the text discusses the problem "is $L(M)$ regular?", which it calls $REGULAR_{TM}$. The text reduces A_{TM} to it by the first reduction at left below, except using $\{0^n 1^n\}$ rather than PAL as the resulting language of M' when M accepts w:



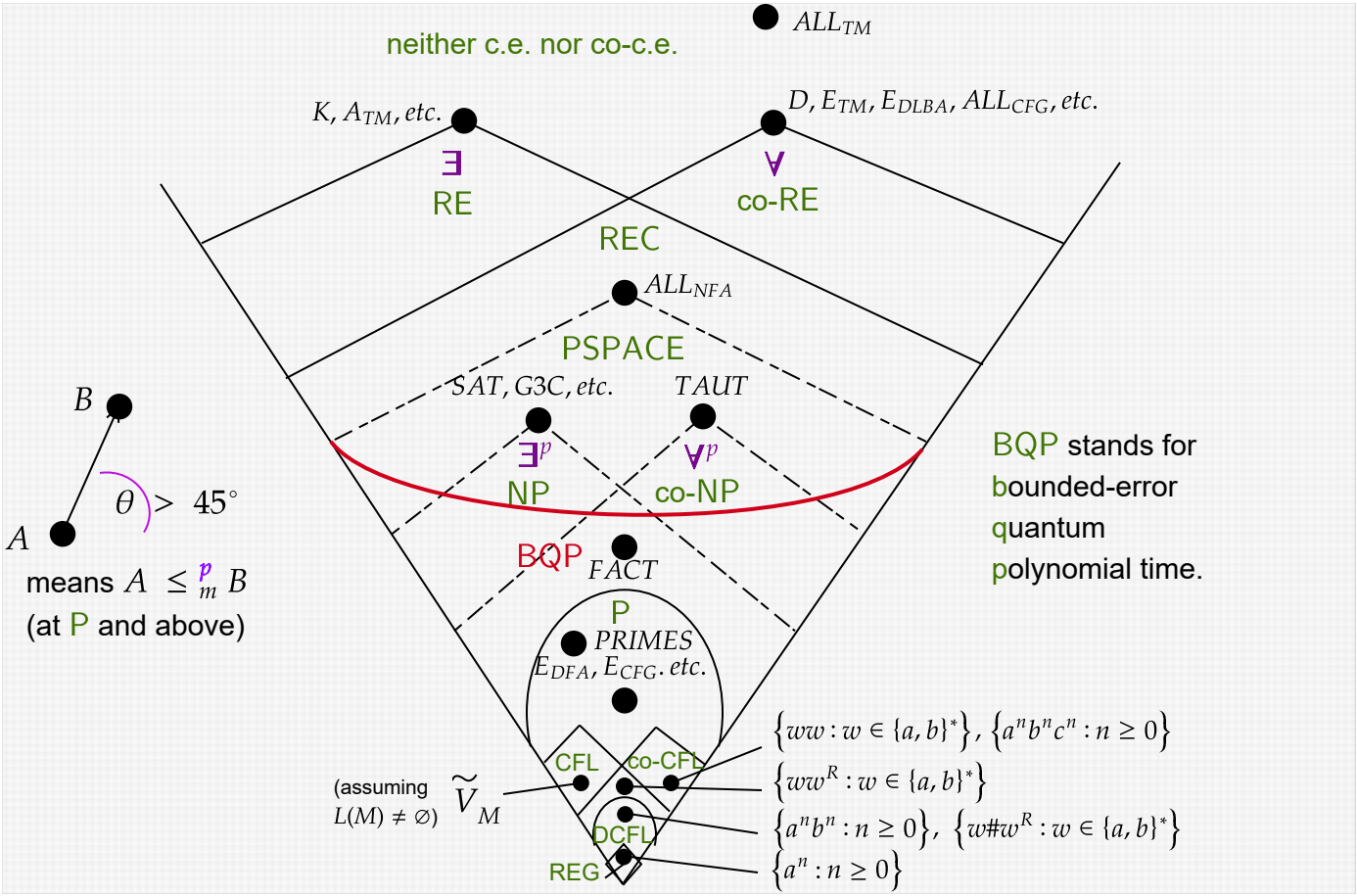
But we can also reduce D_{TM} to it via the reduction at right (as well as by modifying the "delay switch" to check for palindromes on the "panic" branch). So $REGULAR_{TM}$ is neither c.e. nor co-c.e. In fact, $REGULAR_{TM}$ is a notch higher in the "degrees of undecidability" pecking order than ALL_{TM}.

How about for P versus NP? Here we do not have clear principles:

1. We do not have a notion of what behavioral features of programs are *essential* for polynomial-time computation.
2. We **do** know that reasoning about extensional properties, or about input-output behavior and timing alone, **cannot** resolve P versus NP. This is the so-called "Relativization" or "Oracles" barrier, which the text covers in section 9.2.
3. We do not have a measure of how fast programs make progress **internally** toward a goal.
4. The dividing line between "in P" and "NP-hard" is not always sharp. Some cases nestled inamongst hard ones are in P via "Accidental Algorithms" What **is** known is that **few** natural computational problems are still lying in-between P and NP-complete...the Factoring problem is one of them.

Even with the Church-Turing Thesis, things get muddied when we take it down to polynomial time. 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*.

When we add quantum polynomial time---called **BQP**---to our map, it gets even more mixed up:



Maybe AI will solve the great unknown questions about computation? Or AI might just wind up exemplifying them...