

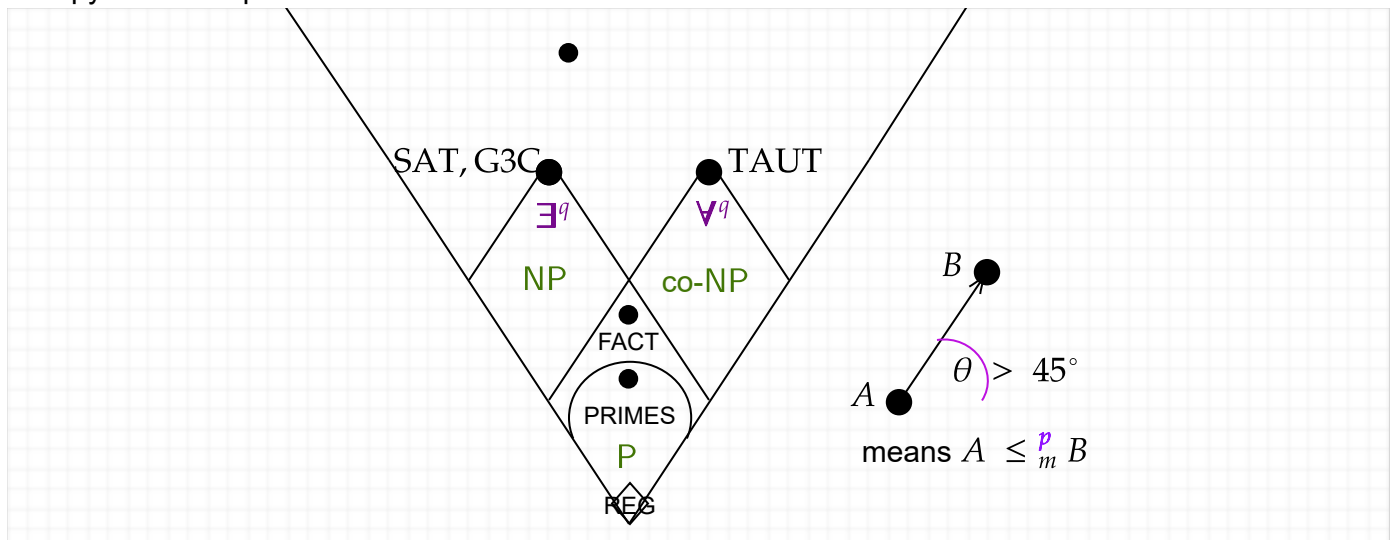
CSE491/596 Lecture Mon. 10/16/23: NP-Completeness and Cook-Levin Theorem

To make a separate definition: A language B is **NP-complete** if:

- $B \in \text{NP}$
- For all languages $A \in \text{NP}$, $A \leq_m^p B$.

The second condition by itself defines B being **NP-hard**.

If B and B' are both NP-complete, then we have both $B \leq_m^p B'$ and $B' \leq_m^p B$, which we summarize by writing $B \equiv_m^p B'$ and saying that B and B' are **polynomial-time many-one equivalent**. The "steep angle means reduction" convention of the "cone diagram" makes equivalent problems/languages occupy the same point:



Before tackling Cook's Theorem on the NP-completeness of SAT, let's see some simpler examples of *reductions*. Consider these decision 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

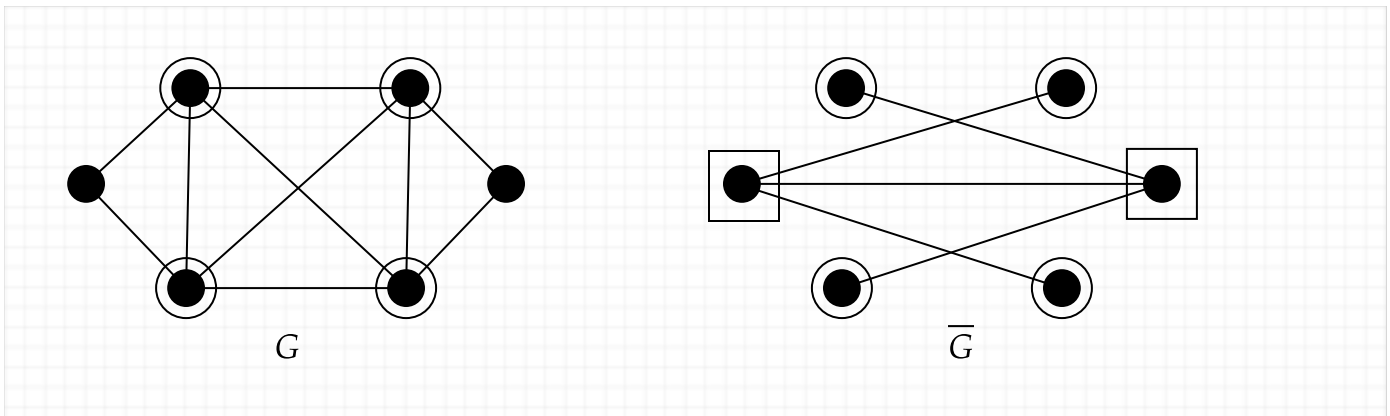
Instance: An undirected graph $G = (V, E)$ and a number $k \geq 1$.

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 ?

Important to keep straight: 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. An important point is that with $n = |V|$, there are 2^n subsets S that might have to be considered. A polynomial-time algorithm cannot try each one. Within S , however, there are at most n^2

pairs (u, v) that have to be considered. Those can all be iterated through to check the body of the condition in quadratic time, so it becomes a polynomial-time decidable predicate $R(G, S)$. It is not even true that this predicate gets negated between the two languages, because it includes the "for each" part. It is because this runs over only polynomially-many pairs that I suggest the convention of saying "for each" rather than "for all" there. What actually gets complemented *is the graph G* , as expressed by this fact:

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



Therefore, the simple reduction function $f(G, k) = (\bar{G}, k)$ reduces **CLIQUE** to **IND SET** and also vice-versa, so the problems are \equiv_m^p equivalent. [Note that this skips writing the angle brackets around $\langle G, k \rangle$; by now that's AOK.] 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 misleadingly, 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 ?

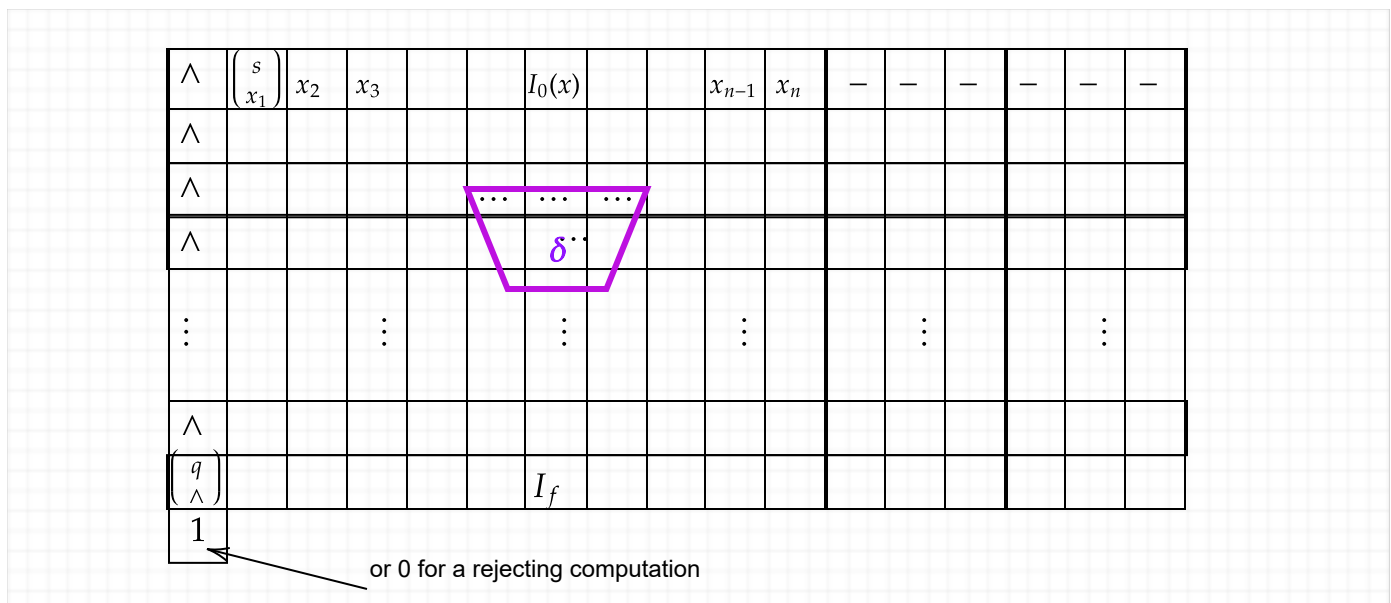
Then **IND SET** and **VC** reduce to each other via the reduction $g(G, k) = (G, n - k)$ (where it is understood that $G = (V, E)$ and $n = |V|$.)

Well, we've shown that these problems are mapping-equivalent to each other, but we haven't shown that they are **NP-hard** yet. We will first show that **SAT** is **NP-hard**.

The Cook-Levin Theorem

The central theorem that SAT is NP-complete used to be credited only to Stephen Cook of the University of Toronto in late-1970, early 1971. (He was born in Buffalo and grew up nearby.) Now we recognize Leonid Levin as having come up with it independently in the Soviet Union, even though he did not publish until 1973. Part reason is that Levin gave a technically stronger result from a more cryptography-minded train of thought. Levin is now at Boston University.

Before we state and prove the theorem, let us see one more application of the idea of tracing a sequence of IDs $I_0(x), I_1, I_2, \dots, I_t$ that represent a valid t -step computation by a TM M , in this case a DTM. Whereas the Kleene T -predicate pictures them side-by-side, now we will stack them up into $t + 1$ columns in a grid. For visual convenience we will suppose M is a 1-tape TM whose tape has a left end and is infinite only to the right, but this is not essential and we could add another grid to handle a second tape, with wires between the grids as well as within them. But for polynomial time, the simple one-plane grid is enough. Initially it has $n + 1$ columns to hold the \wedge left-endmarker and the input x . Over t steps, M cannot possibly visit more than t more cells, so we can lay the whole thing out on a $(t + 1) \times s$ grid with $s \leq t + 1$.



Every cell contains either a character in the work alphabet Γ of M or a pair in $Q \times \Gamma$ of a state and a char. We can use a binary encoding (a-la ASCII) of both. Then we can program a fixed finite function in Boolean logic, depending only on the instructions δ of M , that determines the contents of a cell in any row $i \geq 1$ depending only on the contents of it and its neighbor cell(s) in row $i - 1$ for the previous timestep. The top row is initialized to $I_0(x)$ plus blanks to fill out the remaining columns up to t .

Because NAND is a universal gate, we can program the entire grid into a Boolean circuit C_x entirely of NAND gates, with an output wire w_0 at the bottom giving the final results, 1 or 0. Because the formula

for δ over every cell is the same, the circuit C_x has such a regular structure (pun quasi-intended) that it is easily computed in $O(t^2)$ time given x . [Added afterward] The " x " is used only once and the values of its bits do not affect the layout, so we can give it via n **input gates** to what is otherwise a circuit C_n that depends only on the length n of x . We could suppose $\Sigma = \{0, 1\}$ so x is already in binary, but we could also regard the Boolean encoding of $\Gamma \cup (Q \times \Gamma)$ that the circuit is already using as implicit at the inputs, so there are really $n' = O(n)$ binary input gates. The theorem we have proved has its own significance:

Theorem (often attributed to John E. Savage): For any language A in P and all n we can compute in $n^{O(1)}$ time a circuit C_n of NAND gates such that for all $x \in \Sigma^n, x \in A \iff C_n(x) = 1$. \square

The meaning of this theorem is that "software can be burned into hardware." The fact that $f(x) = \langle C_n, x \rangle$ is polynomial-time computable goes into saying that the sequence $[C_n]_{n=1}^\infty$ of circuits is **P-uniform**. The only reason f is not a "regular reduction" just like the reduction to A_{TM} is that C_n needs counting up to $n = |x|$ and more, and FSTs like DFAs cannot do unbounded counting. But it is close-to-regular in other senses of the above "**Scholium**" that in fact we get the stronger notion of being **DLOGTIME-uniform**.

Similar diagram from the ALR notes, ch. 27, section 3, showing how each cell depends on its 3 neighbors in the previous row:

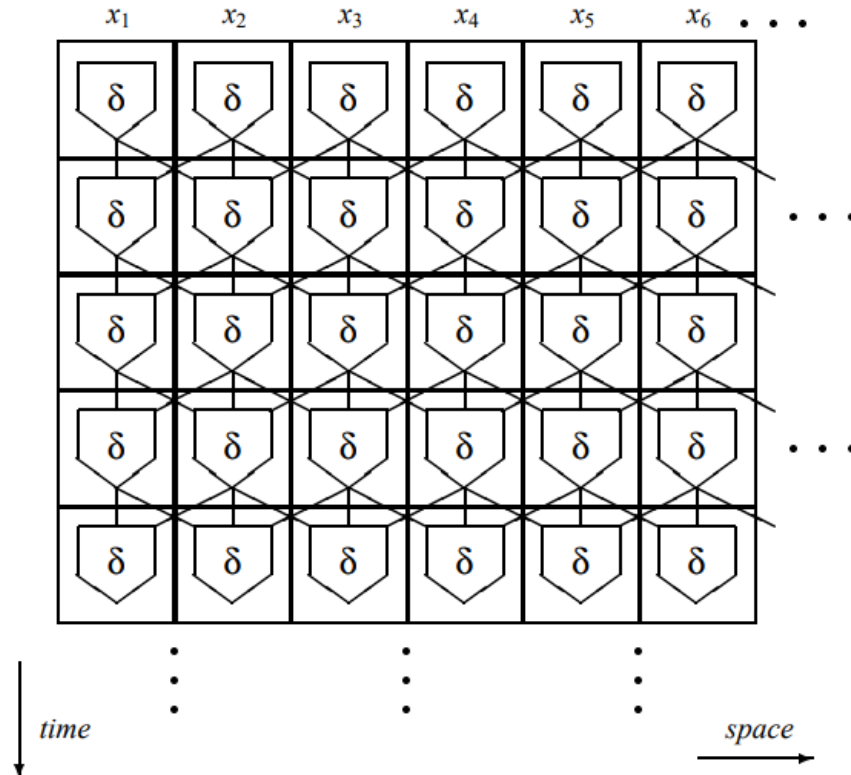


Figure 1: Conversion from Turing machine to Boolean circuits

Statement and Proof of the Cook-Levin Theorem

The reduction goes not only to **SAT** but to a highly restricted subcase of **SAT**:

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 \bar{x}_i . The formula is in **k-CNF** if each clause has at most k distinct literals, *strictly* so if each has exactly k . A 3CNF formula that we will use as a running example is $\phi = (x_1 \vee \bar{x}_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_3 \vee \bar{x}_4)$.

3SAT

Instance: A Boolean formula $\phi(x_1, \dots, x_n) = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ in 3CNF.

Question: **Is there an** assignment $\vec{a} = a_1 a_2 \cdots a_n \in \{0, 1\}^n$ such that $\phi(a_1, \dots, a_n) = 1$?

Of course **3SAT** \leq_m^p **SAT** because it is a restricted case of **SAT**. Cook actually showed **SAT** \leq_m^p **3SAT** in general, but that gets swept up anyway in the proof.

Theorem [Cook 1971, Levin 1971--73]: **3SAT** is **NP**-complete under \leq_m^p , where the reduction function also yields an efficient 1-to-1 correspondence between satisfying assignments and witnesses for the source problem.

The following "circuit-based" proof is actually by Claus-Peter Schnorr from 1978.

Proof. We have already seen that **SAT** is in **NP** and verifying **3SAT** is even easier---see notes below. Now let any $A \in \text{NP}$ 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 ,

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

and such that V_R runs in time $p(r)$ where $r = n + q(n)$. Earlier we stated " $|y| \leq q(n)$ " as the bound on witnesses, but now we are entitled to "play a trump card" by saying that the encoding scheme used to define $\langle x, y \rangle$ first puts things entirely in binary notation with the y parts padded out to the exact length $q(n)$. Since whatever alphabet A was originally defined over can be binary-encoded with only a constant-factor expansion of length, we can regard the length n as meaning *after* the encoding is applied. Since the reduction function f we are building is given x , its length n is a known quantity, so we can finally specify $\langle x, y \rangle$ as just being the concatenation xy of the binary strings. Then $|\langle x, y \rangle|$ really does equal $n + q(n)$. (We abbreviate $q(n)$ as just q .)

Now we apply Savage's theorem to V_R . For each n , we get a circuit C_n with $n + q$ input gates, the

first n for the bits x_1, \dots, x_n of (the binary encoding of) x , and the others for y_1, \dots, y_q , such that $C_n(xy) = 1 \iff V_R$ accepts $\langle x, y \rangle$. Since NAND is a universal gate, we may suppose every gate in the body of C_n is NAND. Since V_R runs in time $p(r)$, the size of C_n is order-of $p(r)^2 = p(n + q(n))^2$. Moreover, because C_n has such a regular structure, we have:

- the function $f_0(x) = \langle C_{|x|} \rangle$ is computable in $p(n + q(n))^2$ time, which is polynomial in n , and
- C_n itself depends only on $n = |x|$, not on the values of the bits of x .

Now we build a Boolean formula ϕ_n out of C_n . After the above window-dressing, this comes real quick. We first allocate variables x_1, \dots, x_n and y_1, \dots, y_q to stand for the input gates, so that the positive literal x_i is carried by every *wire* out of the gate x_i , and likewise every wire out of the gate y_j carries y_j . Then we allocate variables w_0, w_1, \dots, w_s for every other wire in the circuit, where w_0 is the output wire and $s = O(p(n + q)^2)$ is also proportional to the number of NAND gates g , since every NAND gate has exactly two input wires. Then every evaluation of C_n carries a Boolean value through each wire and so gives a legal assignment to these variables---but not every assignment to the wire variables is a legal evaluation of the circuit. If it is not legal, then it must be inconsistent at some NAND gate. We write ϕ_n to enforce that all gates work correctly.

So consider any NAND gate g in the circuit, calling its input wires u and v , and consider any output wire w (there will generally be more than one of those) from g . Define

$$\phi_g = (u \vee w) \wedge (v \vee w) \wedge (\bar{u} \vee \bar{v} \vee \bar{w}).$$

Note this is in (non-strict) 3CNF where the literals in each clause have the same sign. The point is that ϕ_g is satisfied by, and only by, the assignments in $\{0, 1\}^3$ that make $w = u$ NAND v . We can't have u, v, w all be true, and if u or v is false, then w must be true. Thus an assignment to all the variables satisfies ϕ_g if and only if it makes the gate g work correctly for the output wire w . So:

$$\phi_n = \bigwedge_g \phi_g$$

is a (non-strict) 3CNF formula that is satisfied by exactly those assignments that are legal evaluations of C_n . We will finally get the effect of "searching for" a witness y to the particular x by fixing the x_i variables to the values given by the actual bits of x and mandating that $w_0 = 1$. This is all done by the "singleton clauses" (w_0) and for $1 \leq i \leq n$,

$$\beta_i = (x_i) \text{ if the } i\text{-th bit of } x \text{ is } 1, \text{ else } \beta_i = (\bar{x}_i).$$

Thus we finally define the reduction function f by

$$f(x) = \phi_x = \phi_n \wedge (w_0) \wedge \beta_1 \wedge \dots \wedge \beta_n.$$

Then $f(x)$ is computable by one streaming pass over the circuit C_n , and so is computable in the same polynomial $O(p(n) + q(n))^2$ time as C_n . For the mapping of the strings x , we have:

$$x \in A \iff (\exists y : |y| = q(|x|)) C_n(xy) = 1 \iff (\exists y \in \{0, 1\}^q, w \in \{0, 1\}^{s+1}) : \text{the assignment } (x, y, w) \text{ satisfies } \phi_n \wedge w_0 \iff \phi_x \in \text{3SAT.}$$


For the witnesses, the point is that once a y is chosen, on top of x being given (and fixed by the β_i clauses), the values of the rest of the wires in C_n are determined by evaluating all the gates beginning at the top. Hence there is no choice in setting the wire variables w_k besides $w_0 = 1$. Thus the satisfying assignments are in 1-to-1 correspondence with strings y such that $V_R(\langle x, y \rangle) = 1$. (If $x \notin A$ then the correspondence is "none-to-none.") \square

We have abstracted this to a circuit C_n st. $x \in A \iff \exists y \text{ s.t. } C_n(x, y) = 1$

$x_1 - x_2 - \dots - x_n \quad y_1 - \dots - y_j - \dots - y_{p(n)}$

We can stamp out C_n in $O(p(n)^2)$ time.

(u and/or v can be wires from inputs x_i or guesses y_j)



Gate g functions correctly on bits u, v giving output w if and only if ϕ_g is made true, where

$$\phi_g = (u \vee w) \wedge (v \vee w) \wedge (\bar{u} \vee \bar{v} \vee \bar{w}).$$

Final ϕ_x is:

$$\left(\bigwedge_{\text{NAND gates } g \text{ in } C} \phi_g \right) \wedge (w_0) \wedge \text{Singleton clauses } (x_i) \text{ or } (\bar{x}_i) \text{ to set those vars to the actual bits of } x. \text{ E.g. } x = 011 : (\bar{x}_1) \wedge (x_2) \wedge (x_3)$$

Thus ϕ_x is computed by an $\tilde{O}(p(n)^2)$ time function of x , and for all $x \in \Sigma^*$:

$$x \in A \iff (\exists y : |y| \leq p(n)) R(x, y) \iff (\exists y) C_n(x, y) = 1 \iff \exists (y, \bar{w}) \phi_x(x, y, \bar{w}) = 1 \iff \phi_x \in \text{3SAT. } \square$$

wires u, v, w .