NP = Non-deterministic Polynomial Time
is a large family of problems
expect some are <u>not</u> solvable in polynomial time.

NP-hard: Problems at least as hard as <u>everything</u>
in NP.

If can solve any one of them
in poly time, then can solve every NP
problem in poly time.

NP-complete: NP-hard and in NP
essentially equivalent to each other
The hardest problems in NP.

Idea behind NP: (motivation before definition)
Problems for which it might be hard to find the
answer. But once found is easy to check.

Like systems of equations:
takes work to find solution, but easy to check.

Linear equations are in polynomial time,
but Systems of Polynomial Equations are hard.
Abbreviate $\boxed{\text{SPE.}}$

SPE: Have variables, say $x_1, \dots, x_n$,

and polynomials $p_1(x), \dots, p_k(x)$.

Problem 1: find x s.t. $p_i(x) = 0$ for all i?

$\longrightarrow$ Problem 2: Does there exist x s.t. $p_i(x)=0$ for all i?

Problem 1: Given this x, can efficiently check if satisfies all equations. If x is rational, and the coefficients of the polynomials are rational, can check.

i: $x^2-2=0$, $x^p-2=0$ varying p

ii size(x) could be much larger than size(p). is inefficient from perspective of $p_1, \dots, p_k$

iii. What if is no solution?

no solution is an answer.

How would we check that?

We go with Problem 2, which has just yes/no answers.

If "yes", is an x that you can (try to) check.

If "no", there might not be.

Problems with yes/no answers are called decision problems

An NP-complete problem: $\{0,1\}$-SPE

Does there exist $x \in \{0,1\}^n$ s.t. $P_i(x) = 0, \ \forall i$?

Now, the solution can <u>not</u> be big: is in $\{0,1\}^n$.

Can evaluate <u>$P_i(x)$</u> in time polynomial in size(P_i),

so can check x efficiently.

For yes answers and x proving answer is "yes"

$P_i(x) = 0 \quad 1 \le i \le k$, call x a witness, certificate, or proof.

For no answers there does not need to be.

Let $q = (P_1 \dots P_k)$ specify an instance of the problem.

Write $q \boxed{\in}$ $\{0,1\}$-SPE <u>if is valid problem</u> with <u>yes answer</u>.

$Ax = b$ dims don't match

<u>Def</u> A problem Y (like $\{0,1\}$-SPE) is in NP

if $\exists$ a polynomial-time algorithm $\mathcal{A}$ (witness checker)

and constant c governing answer size

such that for all q (problem instances)

if <u>$q \in Y$</u> (valid and yes answer)

other w? $\exists$ <u>w</u> (witness) s.t. $\mathcal{A}(q,w) =$ "yes",

and $\text{size}(w) \le c \cdot \text{size}(q)^c$

$w \in \{0,1\}^n$ s.t. $P_i(w) = 0$ $\forall i$

if $q \notin Y$, $\forall w$ s.t. $\text{size}(w) \le c\,\text{size}(q)^c$

$\mathcal{A}(q,w) =$ "no"

If answer is yes, $\exists w$ convinces $\mathcal{A}$
" " " no, $\mathcal{A}$ will never say "yes"
If q is not valid problem, $\mathcal{A}$ says no
w could be larger than q, but rarely is.
$\text{size}(w) \le \text{poly}(\text{size}(q))$
Time of $\mathcal{A}$ is poly in size of q.

$\{0,1\}$- SPE is in NP

<u>Def.</u> A problem Υ is in P if
$\exists$ a polynomial-time algorithm $\mathcal{A}$ s.t.
$q \in \Upsilon \Rightarrow \mathcal{A}(q) = \text{"yes"}$
$q \notin \Upsilon \Rightarrow \mathcal{A}(q) = \text{"no"}$

$P \subseteq NP$
Linear feasibility $\exists x$ s.t. $Ax \le b$? is in P

Optimization $\to$ Decision:
$\exists x$ s.t. $f(x) \le t$ and $g(x) \le 0$
then search on t
is $x_1 > 0$ $\quad x_1 \le 2 \quad x_1 \le 4 \quad x_1 \le 8 \ldots$
Can learn x by asking about its bits.
All this is in NP, if f and g poly time computable.

"Algorithm" is a bit vague. Program is more precise.
Turing machines formalize this.
We will use logic circuits.

NP contains very hard problems
Factoring.
Break every public-key crypto scheme.
Design anything given specs.
Prove any theorem, as long as proof is not too long.

How do we prove something is NP-hard?

Reductions.
A Karp reduction from Y to Z is a polynomial time algorithm A s.t.

$$q \in Y \quad \text{iff} \quad A(q) \in Z$$

A transforms problem Y into problem Z
Given A and a Ptime algorithm for Z,
can solve Y in Ptime.
If Y is hard, then Z must also be hard.

A Cook reduction from Y to Z is a polynomial time algorithm $\mathcal{A}$ that decides if $q \in Y$
using an Oracle for Z that decides membership in Z
in constant time.

Can solve Y given a subroutine for Z

Y is polynomial-time reducible to Z $\boxed{\text{"}Y \leq_p Z\text{"}}$
if $\exists$ Cook reduction from Y to Z

Karp reductions are Cook reductions,
and always use Karp reductions
(For all known decision problems in NP
with $Y \leq_p Z$, is a Karp reduction)

Z is NP-hard if $\forall Y \in NP$, $Y \leq_p Z$.

Still seems like a lot to show.
NP-complete problems make this manageable.

Z is NP-complete (NPC)
if $Z \in NP$ and Z is NP-hard.

To show X is NP-hard, just show $Z \le_p X$ for some NP-hard Z.

Then $\forall Y \in NP$, $Y \le_p Z$ and $Z \le_p X \Rightarrow Y \le_p X$

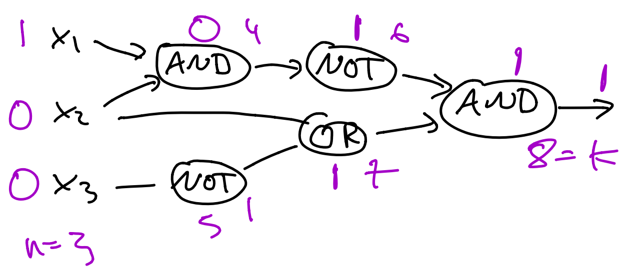
$\Rightarrow X$ is NP-hard

Once we know one such Z, always work from it and never have to write $\forall Y \in NP$ again.

Theorem Circuit-Satisfiability (C-SAT) is NP-complete.

Problem: given a logic circuit with one output, is there an input that makes the output true?

Example



A binary boolean circuit has gates numbered $1, \ldots, k$ s.t. each is either

a. an input (True/False 1/0)

b. the negation (NOT) of a lower numbered gate

c. the AND or OR of two lower numbered gates.

Gate k is the output.

Wolog inputs are gates 1 through n, and $k \ge n$.

Observation: can view every gate g_j as a function of the inputs. If all these equations are satisfied, $y_j = g_j(x_1, \ldots x_n)$, for all j.

$g_j(x_1 \ldots x_n)$

Why C-SAT is NP-Complete:

C-SAT $\in$ NP : given an input can evaluate every gate. ✓

C-SAT is NP-hard because (roughly)

for every algorithm $\mathcal{A}$ that runs in time $T(n)$ on inputs of size n,

For every n there is a circuit C_n with $\leq O((T(n)+n)^2)$ gates s.t.

$C_n(x) = \mathcal{A}(x)$ for all $x \in \{0,1\}^n$

Anything a ptime algorithm can do a polynomial size circuit can, too.

Now, want to prove $\{0,1\}$-SPE is NP-complete and SPE is NP-hard.

We already argued $\{0,1\}$-SPE $\in$ NP.

Thml Now need to prove C-SAT $\leq_p$ $\{0,1\}$-SPE

Let q be an input to C-SAT. That is a circuit.

We need to translate into an instance of $\{0,1\}$-SPE

Let $g_1, \ldots, g_k$ be gates in q

$g_1 \ldots g_n$ are inputs $x_1 \ldots x_n$

Variables will $y_1, \dots, y_k$

$k-n+1$ equations

force $y_j = g_j(y_1, \dots, y_n)$

$g_j = \mathrm{NOT}(g_i)$ $\quad y_j = 1 - y_i \quad \underline{y_j + y_i - 1} = 0$

$g_j = \mathrm{AND}(g_h, g_i)$ $\quad y_j = y_h \cdot y_i \quad y_j - y_h y_i = 0$

$\mathrm{OR}(g_h, g_i)$ $\quad y_j = y_h + y_i - y_h y_i$

For output $y_k = 1$ $\quad y_k - 1 = 0$

If all equs satisfied, and $y_1 \dots y_n \in \{0,1\}$

then $y_k = g_k(y_1 \dots y_n)$

If $\exists x_1 \dots x_n$ s.t. $g_k(x_1 \dots x_k) = 1$

then $\exists y_1 \dots y_n$ that satisfy all equations

$y_j = g_j(x_1 \dots x_n) \quad y_i = x_i \quad 1 \le i \le n$

And conversely if $y_1 \dots y_k$ sat all equs

then $g(y_1 \dots y_n) = 1$, $g \in$ C-SAT

Equations are quadratic and each has at most 3 terms, and coeffs in $\{-1, 0, 1\}$

Thm $\{0,1\}$-SPE $\leq_p$ SPE

so SPE NP-hard

proof same eqns, add $\underbrace{x_i(1-x_i)=0}$, $\forall i$

only if $x_i \in \{0,1\}$.