\( NP = \text{Non-deterministic Polynomial Time} \)

is a large family of problems

expect some are not solvable in polynomial time.

\( NP \)-hard: Problems at least as hard as everything in \( NP \).

If can solve one in polynomial time, can solve everything in \( NP \) in polynomial 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 SPE.
SPE: Have variables, say $x_1, \ldots, x_n,$
and polynomials $p_1(x), \ldots, p_k(x)$.
Problem 1: Find $x$ s.t. $p_i(x) = 0$ for all $i$?
Problem 2: Does there exist $x$ s.t. $p_i(x) = 0$ for all $i$?

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

Issue i: $x^2 - 2 = 0$ does not have a rational solution
Issue ii: What if the solution is much larger than the problem? That is, if $\text{size}(x) \gg \text{size}(p_i, p_m)$?
From the perspective of the problem, this is inefficient.

Issue iii: What if there is no solution?
Is that an answer? How could 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: $\exists$0,1$^k$-SPE

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

Now, the solution can not be big: is in $\{0,1\}^n$.
Can evaluate $P_i(x)$ in time polynomial in $\text{size}(P_i)$, so can check $x$ efficiently.

For yes answers and $x$ proving answer is “yes”

$P_i(x) = 0$ (s.t. $i \leq k$) call $x$ a witness, certificate, or proof.

For no answers there does not need to be.

Let $q = (P_1 \ldots P_k)$ specify the problem.
Write $q \in \exists$0,1$^k$-SPE if it is valid problem with yes answer.

Def A problem $Y$ (like $\exists$0,1$^k$-SPE) is in $NP$

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

and constant $c$ governing answer size

such that for all $q$ (problem instances)

if $q \in Y$ (valid and yes answer)

$\exists w$ (witness) s.t. $A(q, w) = \text{"yes"}$

and size $(w) \leq c \times \text{size}(q)^c$

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

$A(q,w) = \text{"no"}$
Key Points:

If answer is yes, \( \exists w \) that convinces \( A \)

" " " no, \( A \) will not say "yes"

If \( w \) is not valid problem instance, \( A \) says "no"

\( A \) runs in time polynomial in size \( (q) \) and size \( (w) \)

\( w \) could be longer than \( q \) (rarely is)

But size \( (w) \) is polynomial in size \( (q) \)

So time of \( A \) is polynomial in size \( (q) \).

\( \emptyset \) (b) - SPE is in \( NP \)

Def. A problem \( Y \) is in \( P \) if

\( \exists \) a polynomial-time algorithm \( A \) s.t.

\( q \in Y \Rightarrow A(q) = "yes" \)

\( q \notin Y \Rightarrow A(q) = "no" \)

Linear feasibility: \( \exists x \) s.t. \( Ax \leq b \)? is in \( P \)

Can turn optimization problems into decision problems:

ask if \( \exists x \) s.t. \( f(x) \leq t \) and \( g(x) \leq 0 \).

Then do search on \( t \).

Can learn \( x \) by asking about its bits.

All this is in \( NP \).
"Algorithm" is a bit vague. Program is more precise. Turing machines formalize this. We will use logic circuits.

*NP* contains very hard problems

- Factoring
- Breaking any public-key cryptography
- Design anything given specifications.
- Prove any theorem whose 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.

\[ q \in Y \iff A(q) \in Z \]

A transforms problem *Y* into problem *Z*

Given *A* and a *P*-time algorithm for *Z*, can solve *Y* in *P*-time.

If *Y* is hard, then *Z* must also be hard.
A Cook reduction from $Y$ to $Z$ is a polynomial time algorithm $A$ 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 $Z$ "$Y \leq_{p} Z$" 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 $Y \in NP$, $Y \leq_{p} Z$.

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 $Z \leq_p X$ for some NP-hard $Z$.

Then $Y \in \text{NP}$, $Y \leq_p Z$ and $Z \leq_p X \Rightarrow Y \leq_p X$

$\Rightarrow X$ is NP-hard

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

**Theorem** Circuit-Satisfiability ($C$-$\text{SAT}$) 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,.., $k$ st. each is either

a. an input (True or False, 0 or 1)

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 $k > n$. 
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 $j$. 
Why C-SAT is NP-complete:

C-SAT ∈ NP: given an input can evaluate every gate.

C-SAT is NP-hard because (roughly)
for every algorithm A that runs in time T(n) on inputs of size n,

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

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

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

Now, want to prove \( \exists \text{013}-\text{SPE} \) is NP-complete
and \( \text{SPE} \) is NP-hard.

We already argued \( \exists \text{013}-\text{SPE} \in \text{NP} \).

Now need to prove \( \text{C-SAT} \leq_p \exists \text{013}-\text{SPE} \)

Let \( g \) be an input to C-SAT. That is a circuit.

We need to translate into an instance of \( \exists \text{013}-\text{SPE} \)

Let \( g_1, \ldots, g_k \) be the gates in \( g \),
with \( g_{1, \ldots, k} \) being the inputs, \( x_1, \ldots, x_n \)
Our instance of $\text{E}0.13$-SPE will have variables $Y_1, \ldots, Y_K$ and $k-n+1$ equations.

If $g_j = \text{NOT}(g_i)$ we add equation $Y_j = 1 - Y_i$

$g_j = \text{AND}(g_n, g_i)$ " " " $Y_j = Y_n Y_i$

for $Y_n, Y_i \in \text{E}0.13$ this is satisfied iff $Y_j = \text{AND}(Y_n, Y_i)$

$g_j = \text{OR}(g_n, g_i)$, add equation $Y_j = Y_n + Y_i - Y_n Y_i$

To ensure $g_k$ outputs true, add equation $Y_k = 1$

To put these in proper form, rewrite as

\[ \text{NOT} : \ Y_j + Y_i - 1 = 0 \]

\[ \text{AND} : \ Y_j - Y_n Y_i = 0 \]

\[ \text{OR} : \ Y_j - Y_n - Y_i + Y_n Y_i = 0 \]

\[ \text{Output} : \ Y_k - 1 = 0 \]
So, if \( \exists x_1, \ldots, x_n \) s.t. \( g_k(x_1, \ldots, x_n) = 1 \), setting \( y_j = g_j(x_1, \ldots, x_n) \), and \( y_i = x_i \) \( 1 \leq i \leq n \)

results in \( y_1, \ldots, y_n \) satisfying all equations.

Conversely, if \( y_1, \ldots, y_n \) satisfy all equations \( g_k(y_1, \ldots, y_n) = 1 \)

We have reduced any circuit into a very special system of quadratic equations. This sort of system is hard to solve.

**Theorem 2** \( \exists 0(1) \text{-SPE} \leq_p \text{SPE} \)

so \( \text{SPE} \) is \( \text{NP} \)-hard

proof use some equations on \( x_1, \ldots, x_n \), but add \( x_i(1-x_i) = 0 \), \( y_i \)

is satisfied iff \( \forall x_i \in \{0,1\} \)

\( \text{SPE} \) is hard, and probably not in \( \text{NP} \).
(in fact, we know is not).