def P

decision problem $\Pi$ solvable in "poly time" ($\Pi \in P$) if

$\exists$ poly time algorithm $V(.)$ such that

$\forall$ inputs $x$

$\Pi(x) = \text{YES}$ iff $V(x) = \text{TRUE}$

---

def $NP$

decision problem $\Pi$ solvable in "non-deterministic poly-time"

($\Pi \in NP$) if

$\exists$ poly time algorithm $V(.)$

$s.t.$ $\forall$ inputs $x$

$\Pi(x) = \text{YES}$ iff $\exists y$ of size poly($|x|$) s.t.

$V(x, y) = \text{TRUE}$

\[\text{proof/certificate/witness to } x \in L(\Pi) \]

\[\text{verification algorithm} \]

Remark:

$\Pi(x) = \text{NO}$ iff $\forall y$ $V(x, y) = \text{FALSE}$
Polynomial Time Reductions

A, B decision problems

A is poly time reducible to B (denoted $A \leq B$)

if \exists poly time algorithm R such that

- R transforms input $x$ for problem A into $R(x)$ for problem B

- $x$ is yes-input for A iff $R(x)$ is yes-input for B

A instances

\[ \begin{array}{cc}
\text{Yes instances of A} & \text{No instances of A}
\end{array} \]

B instances

\[ \begin{array}{cc}
\text{No instances of B} & \text{Yes instances of B}
\end{array} \]
Why is this a good definition of $A \leq B$?

**Claim** if \( \exists \) p-time algorithm for B
then \( \exists \) p-time algorithm for A

\[
\begin{align*}
X & \xrightarrow{\text{reduction } R} R(x) \\
& \rightarrow \text{Algorithm for } B \\
& \rightarrow \text{Yes/No}
\end{align*}
\]

Algorithm for A

A’s runtime is \( \{\text{runtime on } x\} + \{\text{runtime on } R(x)\} \leq p(|x|) \leq \text{poly } (|x|) \text{ time} \)

Two potential uses:
1) if have algorithm for B, this gives algorithm for A
2) if A is hard, then B must be hard
   \[ \therefore B \text{ is at least as hard as } A \]
**NP-Completeness**

\[ \text{def } D \text{ is NP-Complete if } \]

\[ \cdot D \in \text{NP} \]

\[ \cdot \forall A \in \text{NP}, A \leq D \]

\[ \text{we say } D \text{ is NP-hard} \]

**Question:** How can you show anything is NPC?

must show all decision problems in NP reduce to it

**A first step:**

**Thm [Cook-Levin] 1971**

\[ \text{c-SAT is NP-complete} \]

\[ \text{SAT} \]

\[ \cdot \text{many others follow} \]

\[ \text{Karp 1972} \]
Once you know one...

reductions are transitive!

\[ \text{Thm} \quad D, C \text{ decision problems} \]

\[ \text{if} \quad D \text{ NP-hard} \]

\[ D \leq C \]

\[ \text{then} \quad C \text{ is NP-hard} \]

\[ \text{Pf} \quad \forall A \in \text{NP}, \quad A \leq D \quad \text{under} \quad R_1 \]

\[ \text{but} \quad D \leq C \quad \text{under} \quad R_2 \]

\[ x \xrightarrow{R_1} R_2(R_1(x)) \xrightarrow{R_2(R_1(x)) \in C} \text{Algorithm for } C \]

\[ \text{Algorithm for } A \]

\[ \text{Yes/No} \]

\[ x \in A \iff (R_1(x) \in D) \iff R_2(R_1(x)) \in C \]

\[ R_2 \circ R_1 \text{ is ptime since polynomials compose} \]

Make sure you understand:

- why this reduction has correct behavior (in/out of language)?
- why is it ptime?
To show \( C \) is \( NP \)-complete

1) show \( C \in NP \)

2) show some \( NP \)-complete problem reduces to \( C \)

\[ \{ \text{2 step process} \} \]