NP-complete problems (NPC):

- A subset of NP.

- If any NP-complete problem can be solved in polynomial time, then every problem in NP has a polynomial time solution.

NP-complete languages are the “hardest” languages in NP.

Formal definition of NP-complete languages is based on the concept of polynomial time reducibility.
Examples of problems that belong to P:

1. Find the *shortest* path between two vertices in a directed graph.

2. Does a directed graph have an Euler tour. i.e. a cycle that visits all edges once?

3. Is a Boolean formula in 2-conjunctive normal form satisfiable?
However, their slight variants are in NPC:

1. Find the *longest* path between two vertices in a directed graph.
2. Does a directed graph have a Hamiltonian cycle: a cycle that visits all vertices once?
3. Is a Boolean formula in 3-conjunctive normal form satisfiable?
Polynomial Time Reducibility

**Definition:** A decision problem $A$ is polynomial-time reducible to a decision problem $B$ (written $A \leq_p B$) if:

- There exists a polynomial-time algorithm $F$ that transforms any instance $\alpha$ of $A$ into some instance $\beta = F(\alpha)$ of $B$,

- The answer of $A$ for $\alpha$ is “yes” iff the answer of $B$ for $\beta$ is “yes”.

Design and Analysis of Algorithms: Lecture 26
Polynomial reductions

universe of inputs

inputs for which $P$ yields “no”

inputs for which $P$ yields “yes”

inputs for which $Q$ yields “no”

inputs for which $Q$ yields “yes”

$P$

$Q$

$f$
A Formal Language framework: Reducibility

Every decision problem has a corresponding language = the maximal set of input strings that produce “yes” answers.

Let $L_A, L_B \subseteq \{0, 1\}^*$ be the languages corresponding to the two decision problems A and B, respectively.

**Definition:** $L_A$ is **polynomial-time reducible** to $L_B$ (written $L_A \leq_p L_B$) if:

- there exists a poly-time computable function $f: \{0, 1\}^* \rightarrow \{0, 1\}^*$

- such that for all $\alpha \in \{0, 1\}^*$

$$\alpha \in L_A \text{ if and only if } f(\alpha) \in L_B.$$
Implication of $A \leq_p B$

- $f$ is called a **reduction function** and the poly-time algorithm $F$ that computes $f$ is called a **reduction algorithm**.

- We can use $B$ to solve $A$:
  - Providing an answer to whether $f(\alpha) \in L_B$ directly provides the answer to whether $\alpha \in L_A$. Hence:
    - Solving $A$ is no “harder” than solving $B$. 
Lemma 34.3 If $L_1 \leq_p L_2$ and $L_2 \in P$, then $L_1 \in P$.

- **Proof:**
  - Let $A_2$ be a poly-time algorithm that decides $L_2$.
  - Let $F$ be a poly-time reduction algorithm that does the reduction.
  - We construct a poly-time algorithm $A_1$ that decides $L_1$. 

![Diagram showing the reduction process from $F$ to $A_2$ to $A_1$.]
• **Definition**: $L$ is NP-hard if $\forall L' \in \text{NP}, \ L' \leq_p L$.

• **Definition**: $L$ is NP-Complete if:
  1) $L \in \text{NP}$.
  2) $L \in \text{NP-hard}$.

A very likely possibility:

![Diagram showing relationships between P, NP, NP-Complete, and NP-hard sets]
Circuit Satisfiability problem

A Boolean combinational circuit


\[ X_1 \]
\[ X_2 \]
\[ X_3 \]

OR gate

AND gate

NOT gate
Decision problem

- Is there an assignment to the input that makes the circuit evaluate to TRUE?

\[ \text{CKT-SAT} = \{ \langle \text{CKT} \rangle : \text{CKT has a satisfying assignment} \} \].

- What is the running time of a brute force algorithm?
CKT-SAT is NP-complete

• Lemma 34.5: CKT-SAT $\in$ NP.
  – We can take the number of gates + wires as the size $k$ of the circuit.
  – We can create a binary encoding $\langle \text{CKT} \rangle$ that is polynomial in $k$.
  – Certificate = an assignment of boolean values to the wires.
  – Checking whether the certificate corresponds to a satisfying assignment takes $O(k)$ time.

• Lemma 34.6: CKT-SAT $\in$ NP-hard (pages 1074–1077).
Lemma 34.8: If $L$ is a language such that $L' \leq_p L$ for some $L' \in \text{NPC}$, then $L$ is NP-hard. Moreover, if $L \in \text{NP}$, then $L \in \text{NPC}$.

Proof: Since $L'$ is NP-complete, for all $L'' \in \text{NP}$, we have $L'' \leq_p L'$. By supposition, $L' \leq_p L$, and thus by transitivity, we have $L'' \leq_p L$, which shows that $L$ is NP-hard. If $L \in \text{NP}$, then we also have $L \in \text{NPC}$.

Transitivity: If $L_1 \leq_p L_2$ and $L_3 \leq_p L_3$, then $L_1 \leq_p L_3$ (Exercise 34.3-2).
L ∈ NPC: Generic Proof

- Step 1: prove L ∈ NP.
- Step 2: prove L ∈ NP-hard:
  1. Select a known NP-complete language L’.
  2. Find a reduction algorithm F, s.t. x ∈ L’ ⇔ F(x) ∈ L.
  3. Prove that the algorithm F runs in poly-time.

Up to this point, the only NPC problem we know is CKT-SAT.
Another NPC problem: SAT

Formula satisfiability problem (SAT)

- A instance of SAT is a Boolean formula $\phi$ composed of
  1) $n$ Boolean variables: $x_1, x_2, \ldots, x_n$.
  2) $m$ Boolean connectors: $\land$ (AND), $\lor$ (OR), $\neg$ (NOT), $\to$ (implication), $\leftrightarrow$ (if and only if).
  3) parentheses.

- For example: $\phi = ((x_1 \to x_2) \land (\neg x_1 \lor x_2 \lor x_3)) \to (x_1 \land \neg x_2)$.

- SAT = \{ $\langle \phi \rangle$: $\phi$ has a satisfying assignment (an assignment causes $\phi$ to evaluate to 1) \}.

- For example, $x_1 \lor x_2 \in$ SAT, while $x_1 \land \neg x_1 \notin$ SAT.
Proof:

• Step 1: SAT ∈ NP.
  Certificate is the “truth assignment”. Algorithm merely has to verify, in polynomial time, that the truth assignment produces TRUE.

• Step 2: SAT ∈ NP-hard.
  by proving CKT-SAT ≤ₚ SAT.
The reduction is as follows:

- For each wire $x_i$ in the circuit $C$, the formula $\phi$ has a variable $x_i$.

- For each gate in $C$, make a formula involving the variables of its incident wires that fully describes the behaviour of the gate.
  
  - For example, the operation of the output OR gate (figure on the next page) is $x_5 \leftrightarrow (x_1 \lor x_2)$.

- The formula $\phi$ produced by the reduction is the AND of the circuit-output variable with the conjunction of clauses describing the operation of each gate.
For the above circuit $C$, the formula is

\[ \phi = x_7 \land (\neg x_1 \leftrightarrow x_4) \]
\[ \land (x_5 \leftrightarrow (x_1 \lor x_2)) \]
\[ \land (x_2 \leftrightarrow \neg x_6) \]
\[ \land (x_7 \leftrightarrow (x_4 \land x_5 \land x_3 \land x_6)) \]
SAT: Poly-time reduction

- Easy to see that $C$ is satisfiable $\Leftrightarrow$ $\phi$ is satisfiable.

- The reduction runs in polynomial time.