NP-Complete problems

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**.

From P to NPC

- 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?

From P to NPC

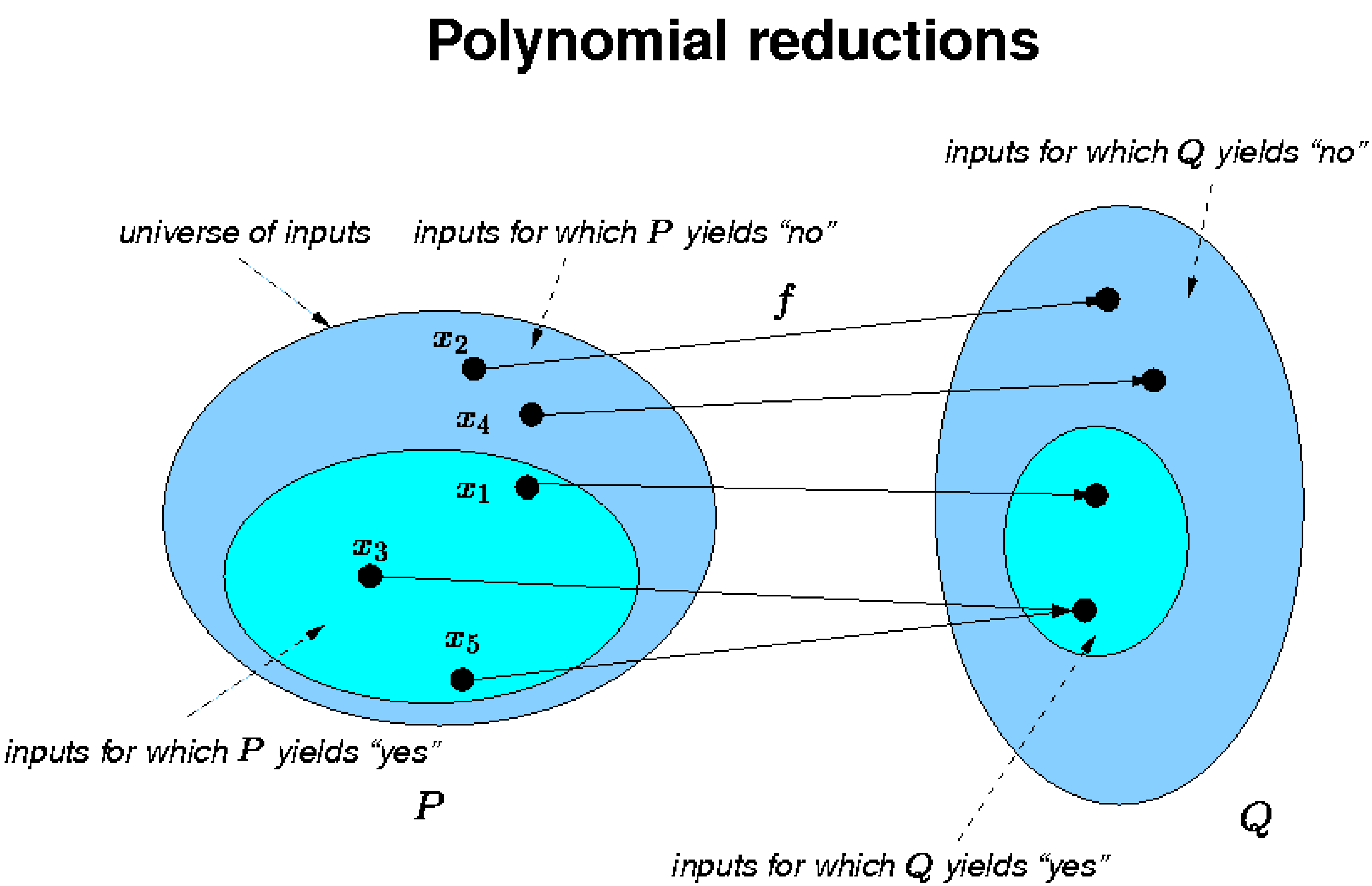
- However, their slight variants are in NPC:
  1. Find the *longest* path between two vertices in a directed graph.
  2. Does a directed graph have a Hamiltonian cycle: a cycle that visits all vertices once?
  3. Is a Boolean formula in 3-conjunctive normal form satisfiable?

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 α of A into some instance $\beta = F(\alpha)$ of B,
- The answer of A for α is "yes" iff the answer of B for β is "yes".

Polynomial reductions



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 $\alpha \in \{0,1\}^*$

  $\alpha \in L_A$ if and only if $f(\alpha) \in L_B$.

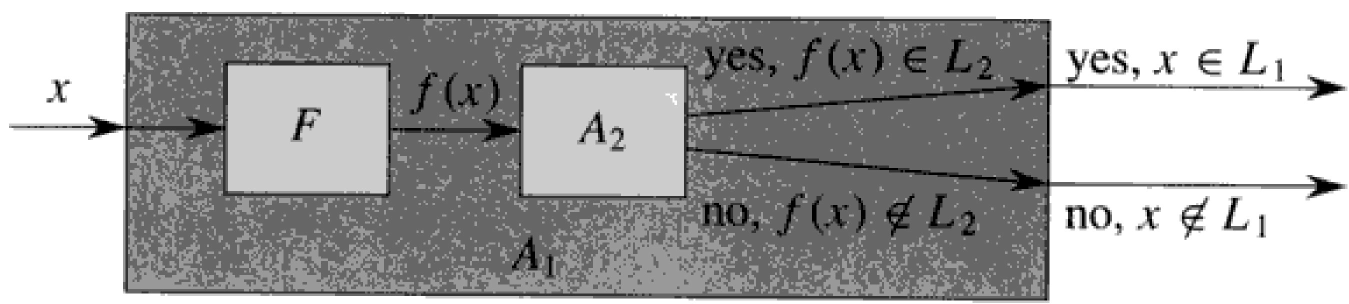
Implication of A $\leq_p$ B

- 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.

Implication of A $\leq_p$ 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.



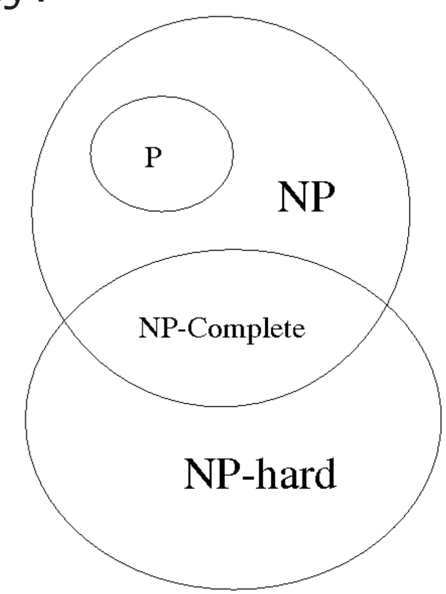
NP-hard and NP-Complete

- **Definition**: L is NP-hard if $\forall$ L' $\in$ NP, L' $\leq_p$ L.
- **Definition**: L is NP-Complete if:

  1) L $\in$ NP.

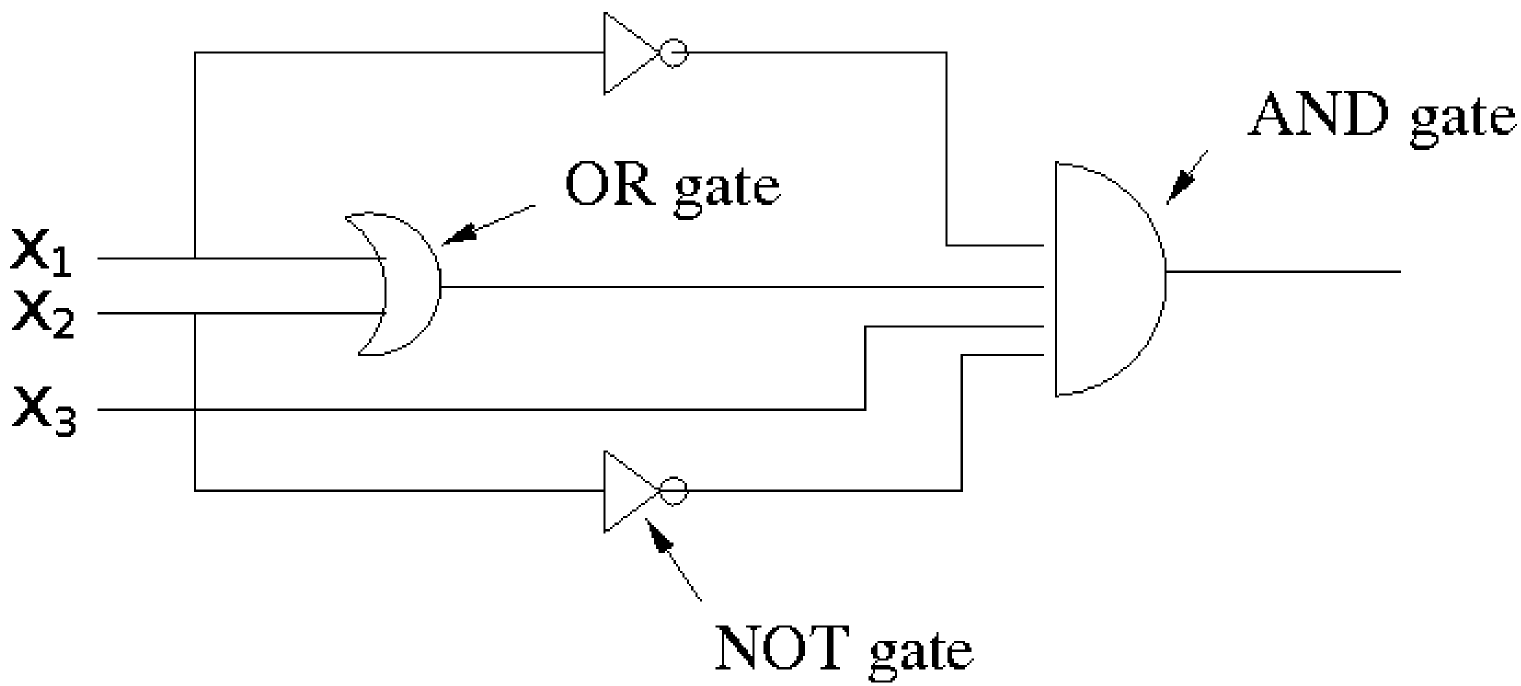
  2) L $\in$ NP-hard.

A very likely possibility:



Circuit Satisfiability problem

A Boolean combinational circuit



CKT-SAT problem

Decision problem

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

  CKT-SAT $= \{\langle \text{CKT} \rangle$: 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 $\langle$CKT$\rangle$ that is polynomial in k.
  - Certificate = an assignment of boolean values to the wires.
  - Checking whether the certificate corresponds to a satisfying assigment takes $O(k)$ time.
- Lemma 34.6: CKT-SAT $\in$ NP-hard (pages 1074–1077).

Alternative definition of NP-completeness

Lemma 34.8: If L is a language such that L' $\leq_p$ L for some L' $\in$ NPC, then L is NP-hard. Moreover, if L $\in$ NP, then L $\in$ NPC.

Proof: Since L' is NP-complete, for all L'' $\in$ 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 $\in$ NP, then we also have L $\in$ 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 $\in$ NPC: Generic Proof

- Step 1: prove L $\in$ NP.
- Step 2: prove L $\in$ NP-hard:
  1. Select a known NP-complete language L'.
  2. Find a reduction algorithm F, s.t. x $\in$ L' $\Leftrightarrow$ F(x) $\in$ 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 ϕ composed of
  1) n Boolean variables: $x_1, x_2, ..., x_n$.
  2) m Boolean connectors: $\wedge$ (AND), $\vee$ (OR), $\neg$ (NOT), $\rightarrow$ (implication), $\leftrightarrow$ (if and only if).
  3) parentheses.
- For example: $\phi = ((x_1 \rightarrow x_2) \wedge (\neg x_1 \vee x_2 \vee x_3)) \rightarrow (x_1 \wedge \neg x_2)$.
- SAT $= \{\langle\phi\rangle$: ϕ has a satisfying assignment (an assignment causes ϕ to evaluate to 1)$\}$.
- For example, $x_1 \vee x_2 \in$ SAT, while $x_1 \wedge \neg x_1 \notin$ SAT.

SAT is NP-Complete

Proof:

- Step 1: SAT $\in$ NP.

  Certificate is the "truth assignment". Algorithm merely has to verify, in polynomial time, that the truth assignment produces TRUE.

- Step 2: SAT $\in$ NP-hard.

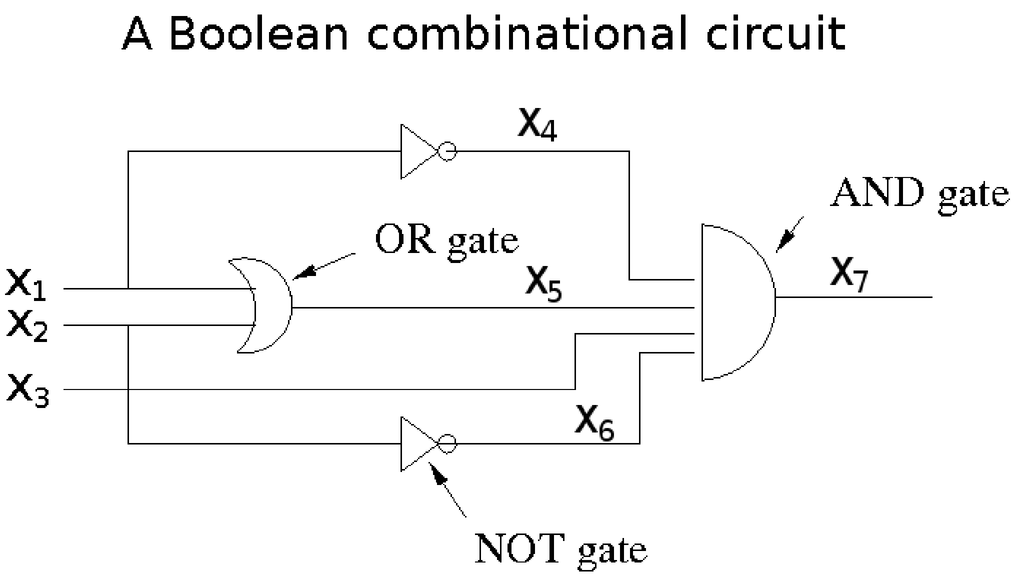
  by proving CKT-SAT $\leq_p$ SAT.

SAT: Poly-time reduction

The reduction is as follows:

- For each wire x_i in the circuit C, the formula ϕ 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 $x_5 \leftrightarrow (x_1 \vee x_2)$.
- The formula ϕ produced by the reduction is the AND of the circuit-output variable with the conjunction of clauses describing the operation of each gate.

SAT: Poly-time reduction



- For the above circuit C, the formula is

$$\begin{aligned}\phi = x_7 & \wedge(\neg x_1 \leftrightarrow x_4) \\ & \wedge(x_5 \leftrightarrow (x_1 \vee x_2)) \\ & \wedge(x_2 \leftrightarrow \neg x_6) \\ & \wedge(x_7 \leftrightarrow (x_4 \wedge x_5 \wedge x_3 \wedge x_6))\end{aligned}$$

SAT: Poly-time reduction

- Easy to see that C is satisfiable $\Leftrightarrow$ ϕ is satisfiable.
- The reduction runs in polynomial time.