Chapter 34
NP-Completeness

Algorithm Analysis

School of CSEE

Hardness of problems

- All algorithms we have studied so far are *polynomial-time* algorithms : $O(n^k)$ for some constant k.

  : $O(n\lg n)$ is not polynomial, but is bounded by polynomial.
- Tractable : not-so-hard

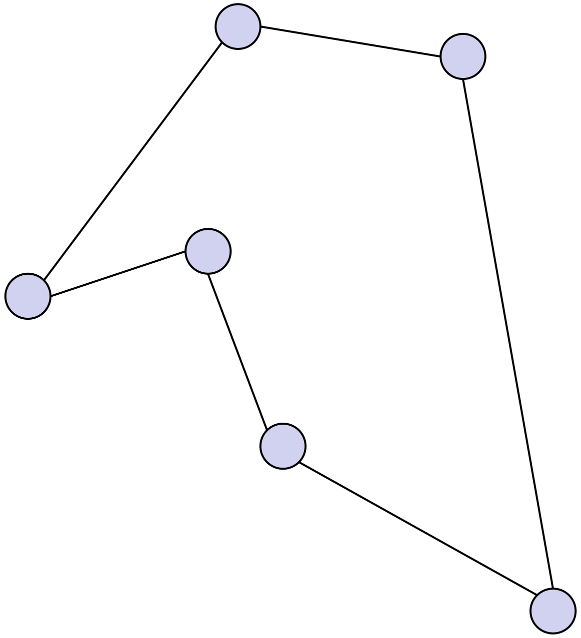
  Intractable : hard, very time consuming
- Problems bounded by polynomial : tractable

  Problems proven to be intractable : intractable

  Problems that have not been proven to be intractable, but polynomial-time algorithm have never been found so far.

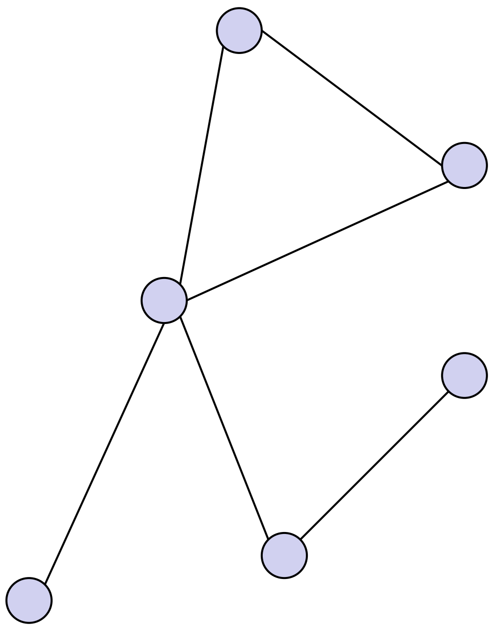
Example: difficult problem

- Traveling Salesperson Problem(TSP)
  - Input : undirected graph with lengths on edges
  - Output : shortest tour that visits each vertex exactly once
- Best known algorithm: $O(n\ 2^n)$ time.



Example: difficult problem

- Clique:
  - Input: undirected graph $G = (V,E)$
  - Output: largest subset C of V such that every pair of vertices in C has an edge between them
- Best known algorithm: $O(n\ 2^n)$ time.



Example: difficult problem

- Knapsack problem
- Satisfiability problem
- Subset sum problem
- Hamiltonian cycle problem
- Bin packing problem
- Job scheduling with penalties problem
- CNF-SAT problem
- Vertex cover problem, etc

The class $\mathbb{P}$

- A problem is called a decision problem if its solution is either ‘*yes*’ or ‘*no*’.
- Let $\mathbb{P}$ denote the class of decision problems that are solvable by algorithms having polynomial (worst-case) complexity.
- If a decision problem is not in $\mathbb{P}$, it will be intractable.

The class $\mathbb{NP}$

- We define a large class of interesting problems, namely $\mathbb{NP}$.
  - Decision problems for which a proposed solution for a given input can be checked in polynomial time to see if it really is a correct solution.
  - Solvable in non-deterministic polynomial time.

A few moments…

- *Optimization problem* : want to find a feasible solution with the best value.
- *Decision problem* : answer is either ‘*yes* (1)’ or ‘*no* (0)’.

- *Deterministic* : same output for same input
- *Nondeterministic* : different output for same input

Traveling salesperson problem

- Optimization problem : Given a complete weighted graph, find a minimum-weight Hamiltonian cycle.

- Decision problem : Given a complete weighted graph and an integer k, is there a Hamiltonian cycle with total weight at most k?

Nondeterministic algorithm

Think of a non-deterministic computer as a computer that magically “guesses” a solution, then has to verify that it is correct.

It has two phases.

- Phase 1 : Nondeterministic ‘guessing’ phase
- Phase 2 : Deterministic verifying phase

Nondeterministic algorithm

- Phase 1 : Nondeterministic ‘guessing’ phase

  Given an instance of a problem, produces some arbitrary solution.

- Phase 2 : Deterministic verifying phase

  Given an instance and proposed solution, verify the solution is correct or not.

Here for same input – instance of problem – phase 1 produces different solution. Thus returns different output in phase 2.

Hamiltonian cycle problem

- Does a graph have a cycle in which every vertex of the graph appears exactly once?
- There does not appear to be a deterministic polynomial time algorithm to recognize those graphs with Hamiltonian cycle.
- There is a simple nondeterministic algorithm:
  - Guess the edges in the cycle and verify that they do indeed form a Hamiltonian cycle.

Nondeterministic algorithm

- The difference between ordinary deterministic algorithm and corresponding nondeterministic algorithm is analogous to the difference between efficiently finding a proof of a statement and efficiently verifying a proof.
  - We intuitively feel that checking a given proof is easier than finding one, but we don't know this for a fact.

$\mathbb{NP}$

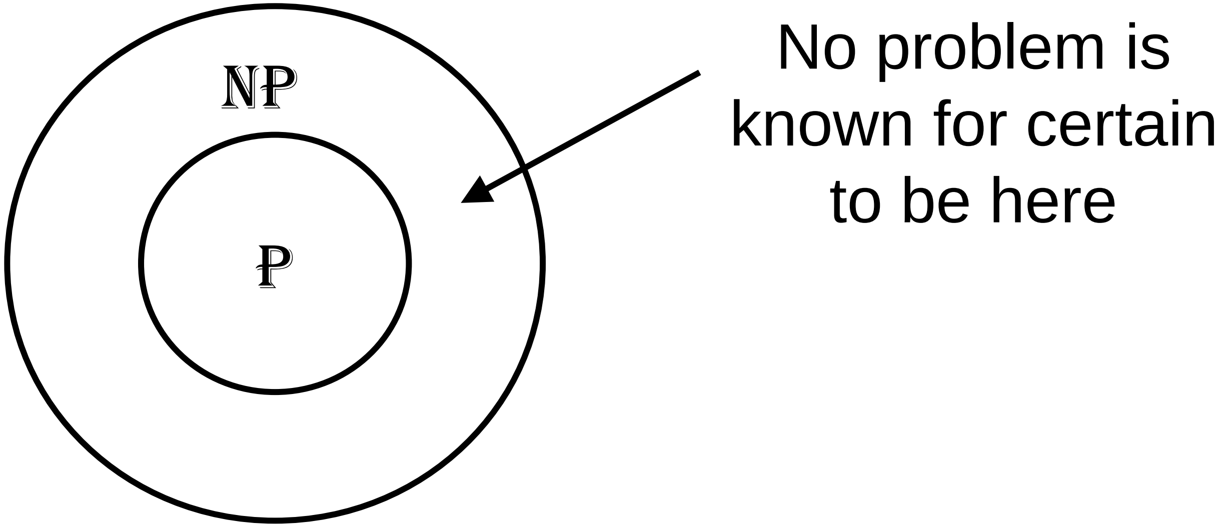
- Decision problem
- There exists a polynomially bounded nondeterministic algorithm : possibility that the problem may have polynomially deterministic algorithm
- $\mathbb{P} \subseteq \mathbb{NP}$: An ordinary deterministic algorithm is a special case of nondeterministic algorithm since it is a phase 2 of nondeterministic algorithm.

No one knows whether $\mathbb{P}=\mathbb{NP}$ or $\mathbb{P} \subset \mathbb{NP}$.

$\mathbb{P}$ and $\mathbb{NP}$

- $\mathbb{P} \subset \mathbb{NP}$

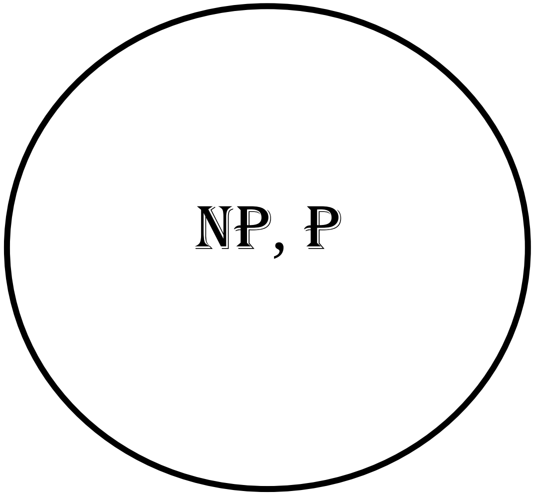
  means that some problems in $\mathbb{NP}$ are intractable.



P and NP

- $\mathbb{P} = \mathbb{NP}$

  means that every problem in $\mathbb{NP}$ are solvable by algorithms having polynomial (worst-case) complexity.



NP-hard & NP-complete

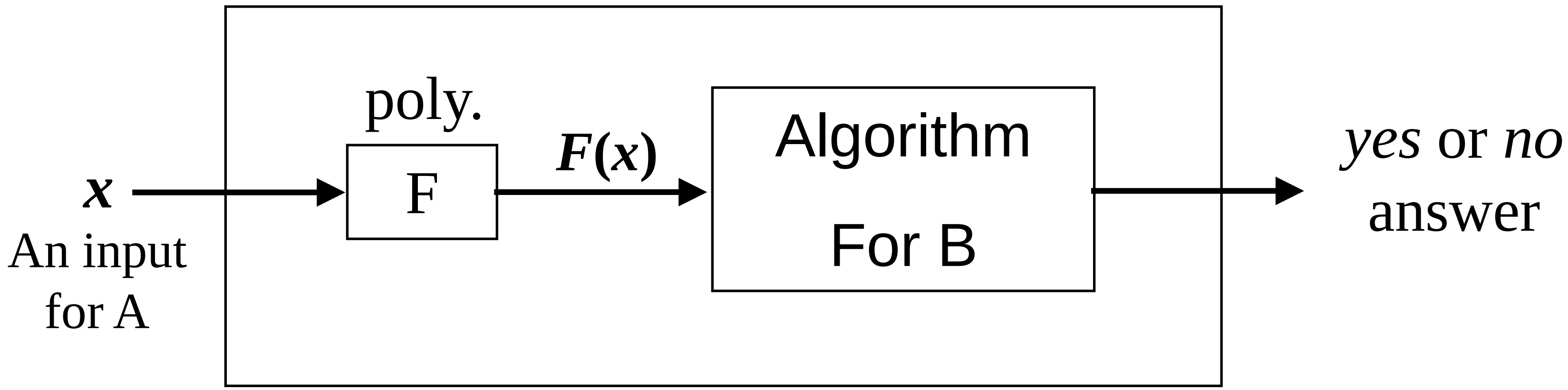
- A problem X is called NP-hard problem if every problem in NP is polynomially reducible to X.
- A problem X is called NP-complete problem if
  1. X belongs to NP, and
  2. X is NP-hard.

If any NP-complete problem is ever proved to belong to P, then P=NP.

Reducibility

- Given two decision problems A and B, we say that A is (polynomially) reducible to B, denoted A ≤p B, if there is a mapping *F* from the inputs (any input instance) to problem A to the inputs (some input instance) to problem B, such that

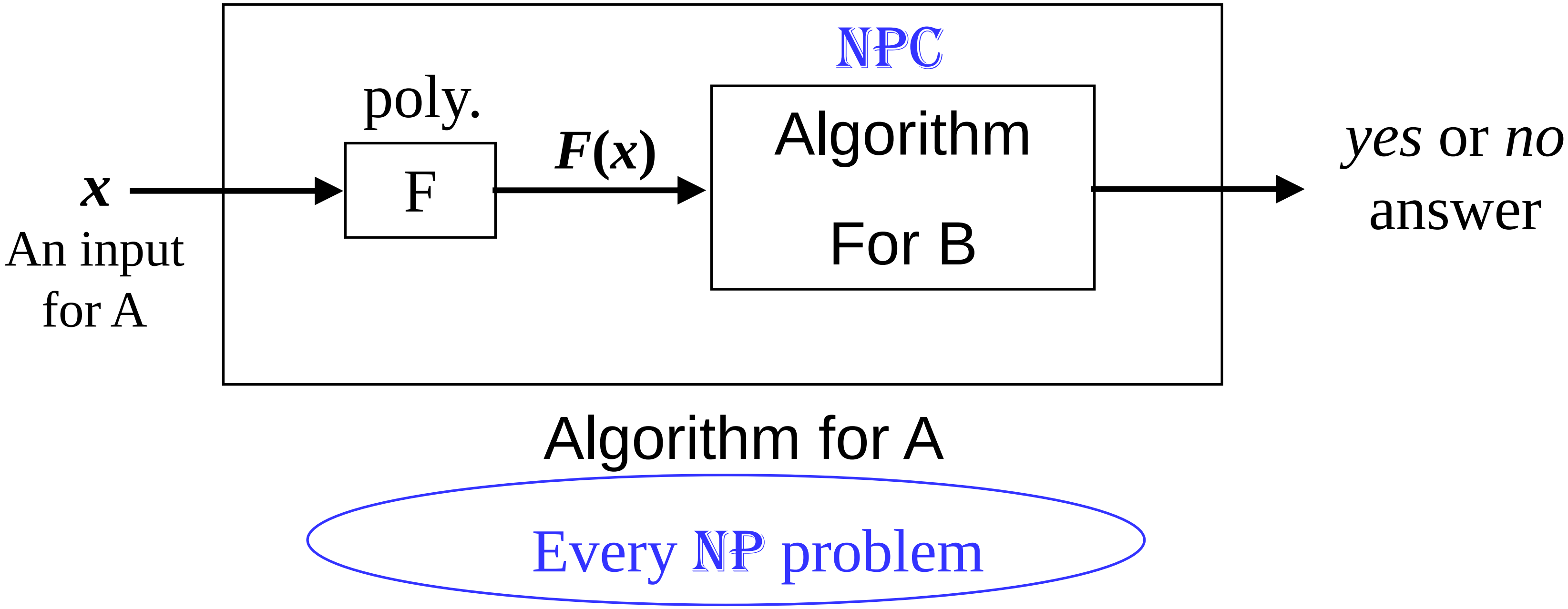
1. *F* can be computed in polynomial time, and
2. the answer to a given input *x* to problem A is *yes* if and only if the answer to the input *F*(*x*) to problem B is *yes*.



Algorithm for A

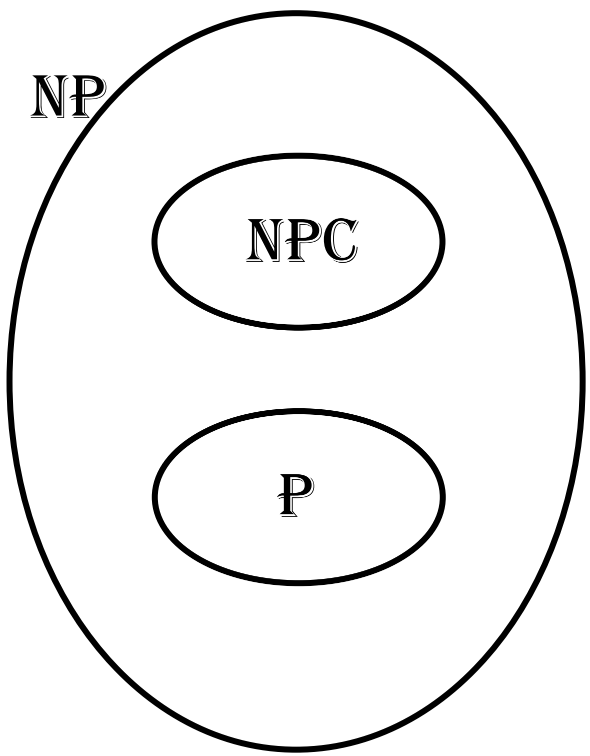
NP-complete

- If A ≤p B and B is in class P, then A is in P.
- Thus, if any NP-complete problem is in class P, then every problem in NP is in P. (P = NP)



P and NP and NPC

If P ⊂ NP,



Suppose P ∩ NPC ≠ 0.

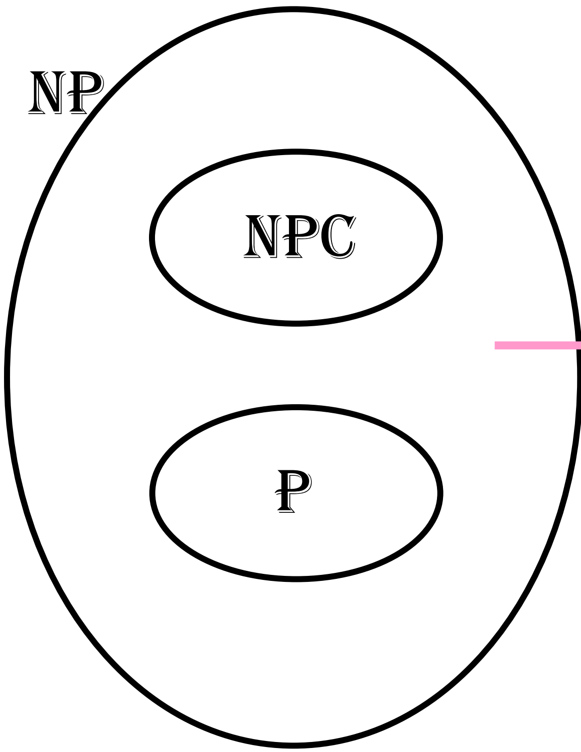
Then it means at least one problem in NPC is in class P.

Thus, if P ∩ NPC ≠ 0, P = NP. (see slide 19.) : Contradiction

Therefore, P ∩ NPC = 0

P and NP and NPC

If P ⊂ NP,

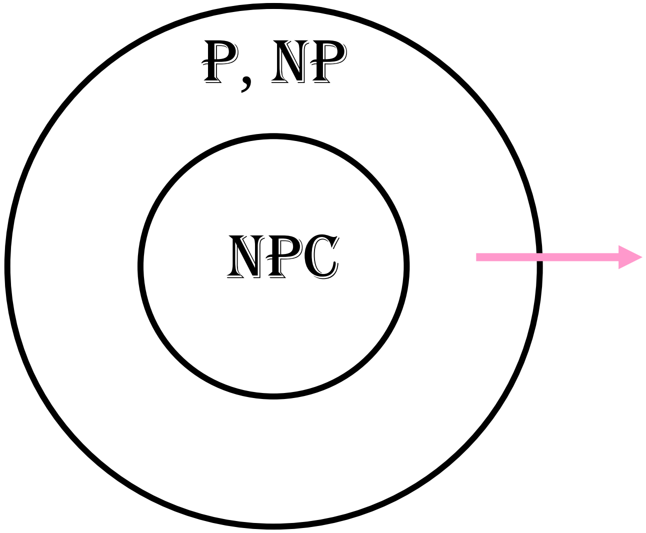


1. If P ≠ NP, it is proved such problems (intractable) must exist. Thus NPC is intractable.

2. But no one has proved that such problem exists.
   Thus, if someone proves such problem exists, then P ≠ NP. (P ⊂ NP).)

P and NP and NPC

If P = NP, NPC problem can be solved in polynomially bounded time.



Trivial decision problem that always answers ‘yes’ (or ‘no’) is in here

CNF-satisfiability problem

- A one-output boolean combinational circuit is satisfiable if there is an input assignment that causes the output of the circuit to be 1.
- CNF-satisfiable problem --- Given a boolean combinational circuit composed of AND, OR, and NOT gates, is it satisfiable?

Cook's theorem

- The CNF-satisfiability problem is $\mathbb{NP}$-complete.

NP-Complete

- Other problems introduced in slides 3 through 5 are proved to be NPC with Cook's theorem.
- To prove problem B in NP is NP-complete, it suffices to prove that some other NP-complete problem is polynomially reducible to problem B.

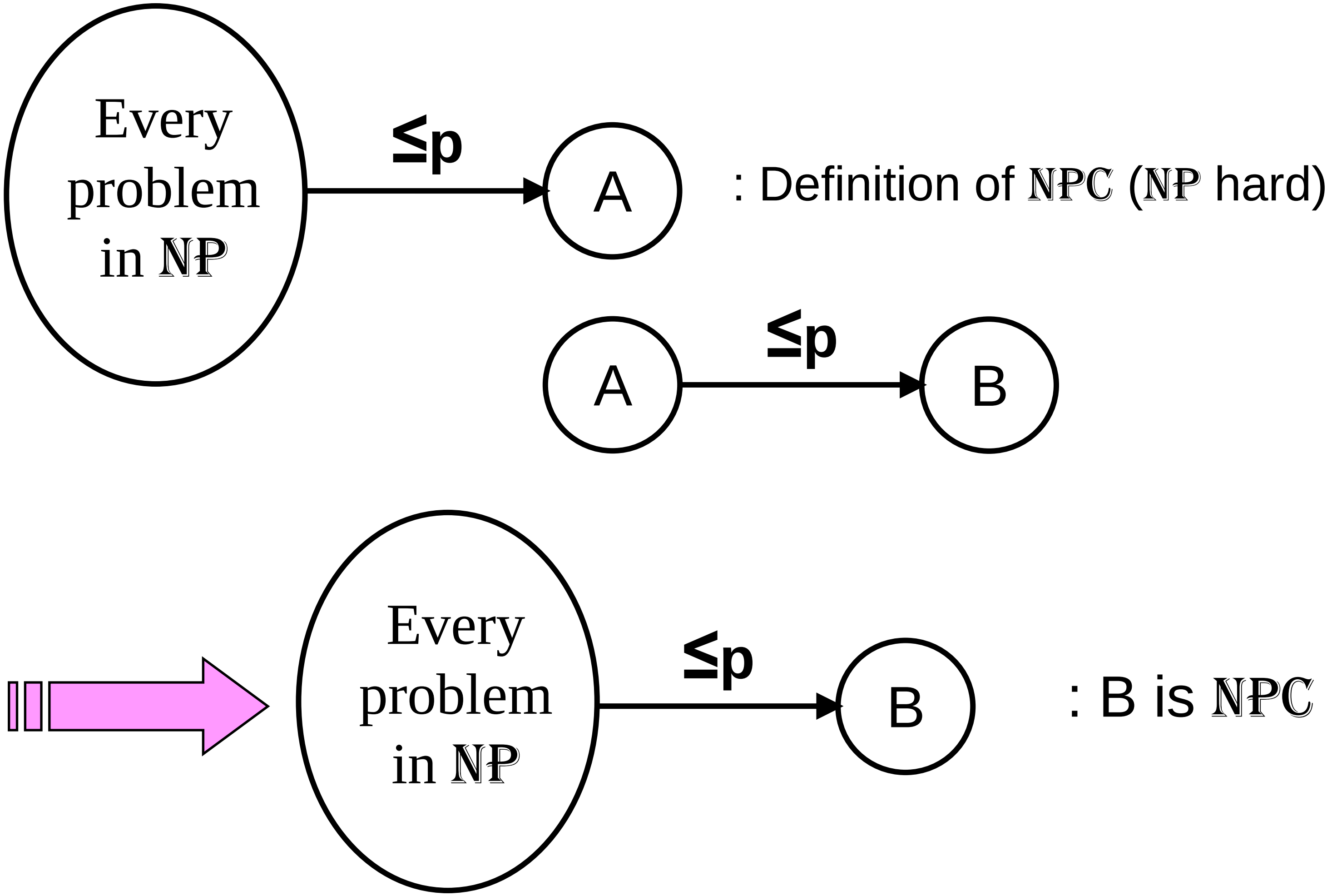
  - Known : Problem A (satisfiability prob.) is NP-complete.

    ➔ every problem in NP is poly. reducible to A.

  - If one proves that A is poly. reducible to B, i.e. A ≤p B, then every problem in NP is poly. reducible to B.

    ➔ B is NP-complete

NP-Complete


Every problem in NP
≤p
A
: Definition of NPC (NP hard)
A
≤p
B
Every problem in NP
≤p
B
: B is NPC

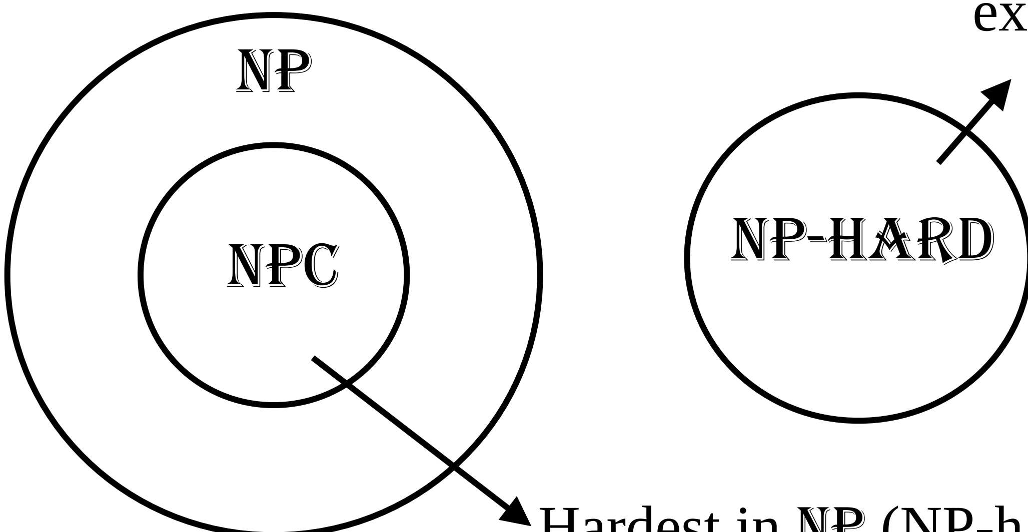
NP-hard & NP-complete

- Optimization problem is as hard as related decision problem, or harder than related decision problem.

  ➔ “at least as hard as”

- Therefore, if related decision problem is NP-complete, then the optimization problem is at least as hard as NP-complete.

As hard as NPC :
ex) related optimization problem



Hardest in NP (NP-hard)

Superpolynomial Algorithm

- Superpolynomial algorithms are computationally infeasible to implement in the worst case, even though we have called a problem tractable. If it has complexity $\Omega(n^k)$ where k is large, such an algorithm may still be computationally infeasible
  - For example, an algorithm having complexity n^{64} will not finish in our lifetime even for n=2.