Let $X \subseteq P$ and $X(s) = 1$.

**Q:** How can Alice convince Bob that $s$ is a yes instance?

**A:** Since $X \subseteq P$, Bob can check whether $X(s) = 1$ by himself, without Alice's help.

Thus, $X \subseteq NP$ and $P \subseteq NP$

Similarly, $P \subseteq Co-NP$, thus $P \subseteq NP \setminus Co-NP$.
Q: How can Alice convince Bob that $s$ is a yes instance?
Let $X \in P$ and $X(s) = 1$

**Q:** How can Alice convince Bob that $s$ is a yes instance?

**A:** Since $X \in P$, Bob can check whether $X(s) = 1$ by himself, without Alice’s help.
Let $X \in P$ and $X(s) = 1$

Q: How can Alice convince Bob that $s$ is a yes instance?

A: Since $X \in P$, Bob can check whether $X(s) = 1$ by himself, without Alice’s help.

The certificate is an empty string
Let $X \in \mathbf{P}$ and $X(s) = 1$

**Q:** How can Alice convince Bob that $s$ is a yes instance?

**A:** Since $X \in \mathbf{P}$, Bob can check whether $X(s) = 1$ by himself, without Alice’s help.

- The certificate is an empty string
- Thus, $X \in \mathbf{NP}$ and $\mathbf{P} \subseteq \mathbf{NP}$
Let $X \in P$ and $X(s) = 1$

**Q:** How can Alice convince Bob that $s$ is a yes instance?

**A:** Since $X \in P$, Bob can check whether $X(s) = 1$ by himself, without Alice's help.

- The certificate is an empty string
- Thus, $X \in NP$ and $P \subseteq NP$
- Similarly, $P \subseteq Co-NP$, thus $P \subseteq NP \cap Co-NP$
Is \( P = NP \)?
Is $P = NP$?

- A famous, big, and fundamental open problem in computer science
- Most researchers believe $P \neq NP$
- It would be too amazing if $P = NP$: if one can check a solution efficiently, then one can find a solution efficiently
Is $P = \text{NP}$?

- A famous, big, and fundamental open problem in computer science
- Little progress has been made
- Most researchers believe $P \neq \text{NP}$
- It would be too amazing if $P = \text{NP}$: if one can check a solution efficiently, then one can find a solution efficiently
Is $P = NP$?

- A famous, big, and fundamental open problem in computer science
- Little progress has been made
- Most researchers believe $P \neq NP$
- It would be too amazing if $P = NP$: if one can check a solution efficiently, then one can find a solution efficiently
- We assume $P \neq NP$ and prove that problems do not have polynomial time algorithms.
Is $P = NP$?

- A famous, big, and fundamental open problem in computer science
- Little progress has been made
- Most researchers believe $P \neq NP$
- It would be too amazing if $P = NP$: if one can check a solution efficiently, then one can find a solution efficiently
- We assume $P \neq NP$ and prove that problems do not have polynomial time algorithms.
- We said it is unlikely that Hamiltonian Cycle can be solved in polynomial time:
  - if $P \neq NP$, then $HC \notin P$
  - $HC \notin P$, unless $P = NP$
Is $NP = Co-NP$?

- Again, a big open problem
Is $\text{NP} = \text{Co-NP}$?

- Again, a big open problem
- Most researchers believe $\text{NP} \neq \text{Co-NP}$. 
Notice that $X \in \text{NP} \iff \overline{X} \in \text{Co-NP}$ and $\text{P} \subseteq \text{NP} \cap \text{Co-NP}$

- People commonly believe we are in the 4th scenario
Outline

1 Some Hard Problems
2 P, NP and Co-NP
3 Polynomial Time Reductions and NP-Completeness
4 NP-Complete Problems
5 Dealing with NP-Hard Problems
6 Summary
**Def.** Given a black box algorithm $A$ that solves a problem $X$, if any instance of a problem $Y$ can be solved using a polynomial number of standard computational steps, plus a polynomial number of calls to $A$, then we say $Y$ is polynomial-time reducible to $X$, denoted as $Y \leq_P X$.
**Def.** Given a black box algorithm $A$ that solves a problem $X$, if any instance of a problem $Y$ can be solved using a polynomial number of standard computational steps, plus a polynomial number of calls to $A$, then we say $Y$ is polynomial-time reducible to $X$, denoted as $Y \leq_P X$.

To prove positive results:

Suppose $Y \leq_P X$. If $X$ can be solved in polynomial time, then $Y$ can be solved in polynomial time.
Polynomial-Time Reductions

**Def.** Given a black box algorithm $A$ that solves a problem $X$, if any instance of a problem $Y$ can be solved using a polynomial number of standard computational steps, plus a polynomial number of calls to $A$, then we say $Y$ is polynomial-time reducible to $X$, denoted as $Y \leq_P X$.

To prove positive results:

Suppose $Y \leq_P X$. If $X$ can be solved in polynomial time, then $Y$ can be solved in polynomial time.

To prove negative results:

Suppose $Y \leq_P X$. If $Y$ cannot be solved in polynomial time, then $X$ cannot be solved in polynomial time.
Polynomial-Time Reduction: Example

Hamiltonian-Path (HP) problem

**Input:** \( G = (V, E) \) and \( s, t \in V \)

**Output:** whether there is a Hamiltonian path from \( s \) to \( t \) in \( G \)
Hamiltonian-Path (HP) problem

**Input:** $G = (V, E)$ and $s, t \in V$

**Output:** whether there is a Hamiltonian path from $s$ to $t$ in $G$

**Lemma** HP $\leq_P$ HC.
Hamiltonian-Path (HP) problem

**Input:** $G = (V, E)$ and $s, t \in V$

**Output:** whether there is a Hamiltonian path from $s$ to $t$ in $G$

**Lemma** $\text{HP} \leq_P \text{HC}$.
Polynomial-Time Reduction: Example

Hamiltonian-Path (HP) problem

**Input:** \( G = (V, E) \) and \( s, t \in V \)

**Output:** whether there is a Hamiltonian path from \( s \) to \( t \) in \( G \)

**Lemma** \( \text{HP} \leq_P \text{HC} \).
Polynomial-Time Reduction: Example

Hamiltonian-Path (HP) problem

**Input:** $G = (V, E)$ and $s, t \in V$

**Output:** whether there is a Hamiltonian path from $s$ to $t$ in $G$

**Lemma** HP $\leq_P$ HC.

**Obs.** $G$ has a HP from $s$ to $t$ if and only if graph on right side has a HC.
NP-Completeness

**Def.** A problem $X$ is called **NP-complete** if

1. $X \in \text{NP}$, and
2. $Y \leq_{\text{P}} X$ for every $Y \in \text{NP}$. 
Def. A problem $X$ is called NP-hard if

1. $X$ is in NP, and
2. $Y \leq_P X$ for every $Y \in NP$.

- NP-hard problems are at least as hard as NP-complete problems (a NP-hard problem is not required to be in NP)
Def. A problem $X$ is called NP-complete if
1. $X \in \text{NP}$, and
2. $Y \leq_{P} X$ for every $Y \in \text{NP}$.

NP-hard problems are at least as hard as NP-complete problems
(a NP-hard problem is not required to be in NP)
NP-Completeness

**Def.** A problem $X$ is called **NP-complete** if

1. $X \in \text{NP}$, and
2. $Y \leq_P X$ for every $Y \in \text{NP}$.

**Theorem** If $X$ is NP-complete and $X \in \text{P}$, then $\text{P} = \text{NP}$.

- NP-hard problems are at least as hard as NP-complete problems (a NP-hard problem is not required to be in NP)
**Def.** A problem $X$ is called **NP-complete** if

1. $X \in \text{NP}$, and
2. $Y \leq_{\text{P}} X$ for every $Y \in \text{NP}$.

**Theorem** If $X$ is NP-complete and $X \in \text{P}$, then $\text{P} = \text{NP}$.

- NP-complete problems are the hardest problems in NP
- NP-hard problems are at least as hard as NP-complete problems (a NP-hard problem is not required to be in NP)
1. Some Hard Problems
2. P, NP and Co-NP
3. Polynomial Time Reductions and NP-Completeness
4. NP-Complete Problems
5. Dealing with NP-Hard Problems
6. Summary
Def. A problem $X$ is called NP-complete if

1. $X \in \text{NP}$, and
2. $Y \leq_p X$ for every $Y \in \text{NP}$.
Def. A problem $X$ is called **NP-complete** if

1. $X \in \text{NP}$, and
2. $Y \leq_p X$ for every $Y \in \text{NP}$.

How can we find a problem $X \in \text{NP}$ such that every problem $Y \in \text{NP}$ is polynomial time reducible to $X$? Are we asking for too much?
A problem $X$ is called **NP-complete** if

1. $X \in \text{NP}$, and
2. $Y \leq_p X$ for every $Y \in \text{NP}$.

How can we find a problem $X \in \text{NP}$ such that every problem $Y \in \text{NP}$ is polynomial time reducible to $X$? Are we asking for too much?

No! There is indeed a large family of natural NP-complete problems.
The First NP-Complete Problem: Circuit-Sat

Circuit Satisfiability (Circuit-Sat)

**Input:** a circuit

**Output:** whether the circuit is satisfiable
key fact: algorithms can be converted to circuits

**Fact** Any algorithm that takes $n$ bits as input and outputs 0/1 with running time $T(n)$ can be converted into a circuit of size $p(T(n))$ for some polynomial function $p(\cdot)$.
key fact: algorithms can be converted to circuits

**Fact** Any algorithm that takes $n$ bits as input and outputs 0/1 with running time $T(n)$ can be converted into a circuit of size $p(T(n))$ for some polynomial function $p(\cdot)$.

Then, we can show that any problem $Y \in \text{NP}$ can be reduced to Circuit-Sat.

We prove $\text{HC} \leq_P \text{Circuit-Sat}$ as an example.
Let \( \text{check-HC}(G, S) \) be the certifier for the Hamiltonian cycle problem: \( \text{check-HC}(G, S) \) returns 1 if \( S \) is a Hamiltonian cycle in \( G \) and 0 otherwise.
Let \( \text{check-HC}(G, S) \) be the certifier for the Hamiltonian cycle problem: \( \text{check-HC}(G, S) \) returns 1 if \( S \) is a Hamiltonian cycle is \( G \) and 0 otherwise.

\( G \) is a yes-instance if and only if there is an \( S \) such that \( \text{check-HC}(G, S) \) returns 1
Let \( \text{check-HC}(G, S) \) be the certifier for the Hamiltonian cycle problem: \( \text{check-HC}(G, S) \) returns 1 if \( S \) is a Hamiltonian cycle in \( G \) and 0 otherwise.

\( G \) is a yes-instance if and only if there is an \( S \) such that \( \text{check-HC}(G, S) \) returns 1.

Construct a circuit \( C' \) for the algorithm check-HC.
HC \leq_P \text{Circuit-Sat}

Let check-HC(G, S) be the certifier for the Hamiltonian cycle problem: check-HC(G, S) returns 1 if S is a Hamiltonian cycle is G and 0 otherwise.

G is a yes-instance if and only if there is an S such that check-HC(G, S) returns 1.

Construct a circuit $C'$ for the algorithm check-HC.

Hard-wire the instance $G$ to the circuit $C'$ to obtain the circuit $C$. 
Let check-HC\((G, S)\) be the certifier for the Hamiltonian cycle problem: check-HC\((G, S)\) returns 1 if \(S\) is a Hamiltonian cycle is \(G\) and 0 otherwise.

\(G\) is a yes-instance if and only if there is an \(S\) such that check-HC\((G, S)\) returns 1

Construct a circuit \(C'\) for the algorithm check-HC

hard-wire the instance \(G\) to the circuit \(C'\) to obtain the circuit \(C\)

\(G\) is a yes-instance if and only if \(C\) is satisfiable
Let check-$Y(s, t)$ be the certifier for problem $Y$: check-$Y(s, t)$ returns 1 if $t$ is a valid certificate for $s$.

$s$ is a yes-instance if and only if there is a $t$ such that check-$Y(s, t)$ returns 1.

Construct a circuit $C'$ for the algorithm check-$Y$.

hard-wire the instance $s$ to the circuit $C'$ to obtain the circuit $C$.

$s$ is a yes-instance if and only if $C$ is satisfiable.
Let check-Y(s, t) be the certifier for problem Y: check-Y(s, t) returns 1 if t is a valid certificate for s.

s is a yes-instance if and only if there is a t such that check-Y(s, t) returns 1

Construct a circuit C′ for the algorithm check-Y

hard-wire the instance s to the circuit C′ to obtain the circuit C

s is a yes-instance if and only if C is satisfiable

Theorem Circuit-Sat is NP-complete.
Reductions of NP-Complete Problems

- Circuit-Sat
  - 3-Sat
    - Clique
    - Ind-Set
      - Vertex-Cover
        - Set-Cover
    - HC
    - 3D-Matching
    - 3-Coloring
    - Knapsack
    - Subset-Sum
    - TSP