CMSC 28100

Introduction to Complexity Theory

Spring 2024
Instructor: William Hoza

The complexity class NP

- Let $L \subseteq \Sigma^*$ be a language
- **Definition:** $L \in \mathrm{NP}$ if there exists a randomized polynomial-time Turing machine M such that for every $w \in \Sigma^*$:
  - If $w \in L$, then $\Pr[M \text{ accepts } w] \neq 0$
  - If $w \notin L$, then $\Pr[M \text{ accepts } w] = 0$
- "<u>N</u>ondeterministic <u>P</u>olynomial-time"

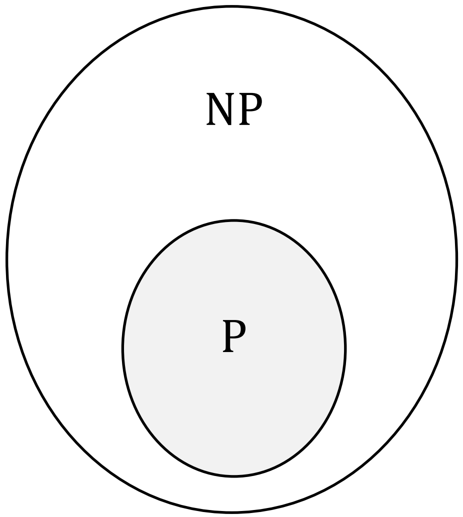
How to interpret NP

- NP is not intended to model the concept of tractability
- A nondeterministic polynomial-time algorithm is not a practical way to solve a problem
- Instead, NP is a conceptual tool for reasoning about computation

“Verification of certificates” perspective

- Let $L \subseteq \Sigma^*$ be a language
- **Claim:** $L \in \mathrm{NP}$ if and only if there exists a deterministic polynomial-time Turing machine V (a “verifier”) and a constant $k \in \mathbb{N}$ such that:
  - For every $w \in L$, there exists a string x (a “certificate” / “witness”) such that $|x| \leq |w|^k$ and V accepts $\langle w, x \rangle$
  - For every $w \notin L$, for every string x, the machine V rejects $\langle w, x \rangle$

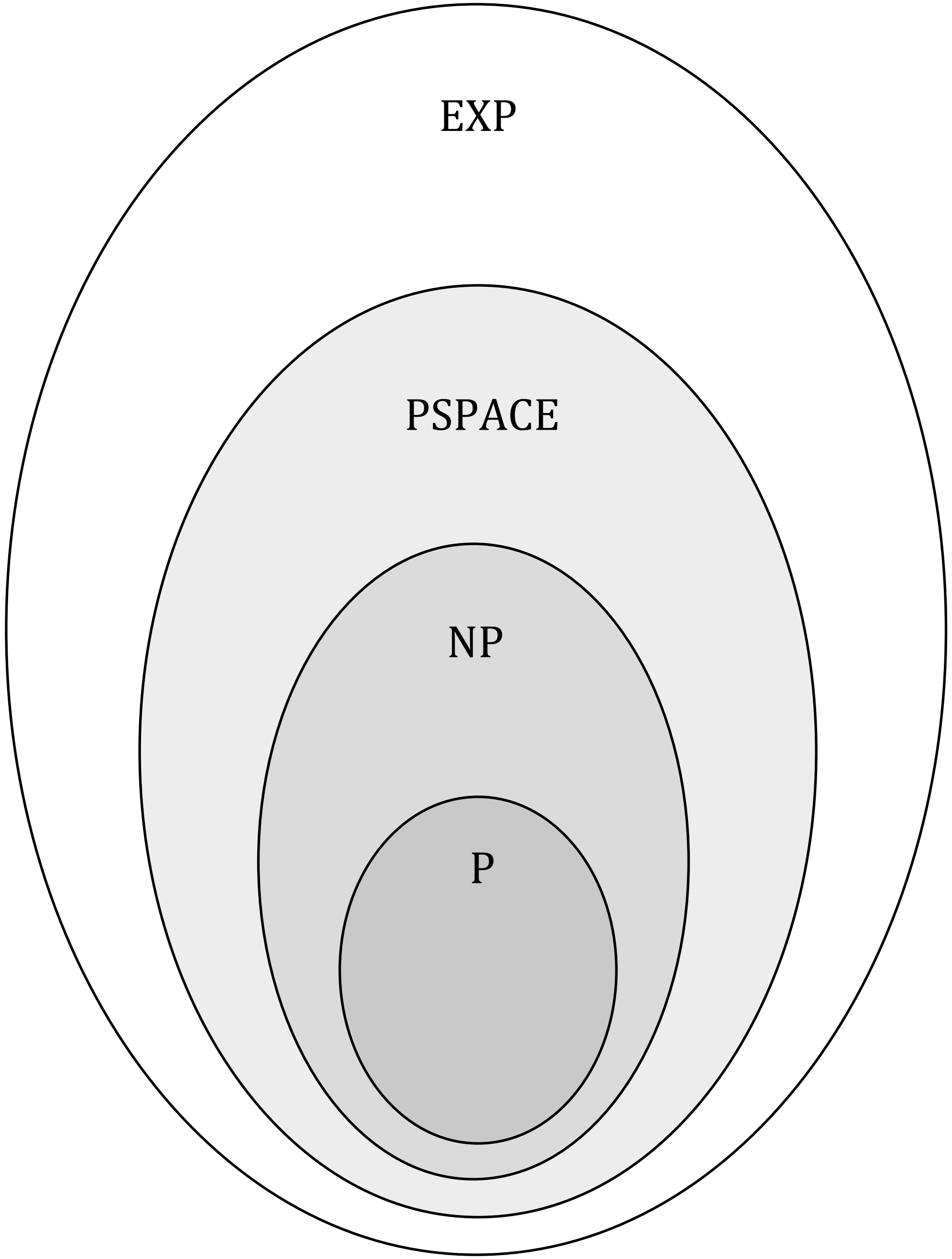
The P vs. NP problem



- $\mathrm{P} \subseteq \mathrm{NP}$
- It is conjectured that $\mathrm{P} \neq \mathrm{NP}$, but nobody knows how to prove it

Solving problems in NP by brute force

- **Claim:** $\mathrm{NP} \subseteq \mathrm{PSPACE}$
- **Proof:** Let M be a nondeterministic TM that runs in time n^k. Given $w \in \Sigma^n$:
  1. For every $x \in \{0,1\}^{n^k}$, simulate M, initialized with w on tape 1 and x on tape 2
  2. If we find some x such that M accepts, accept. Otherwise, reject
- NP can be informally defined as "the set of problems that can be solved by brute-force search"


EXP
PSPACE
NP
P

P vs. NP vs. PSPACE vs. EXP

- $\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE} \subseteq \mathrm{EXP}$
- What we expect: All of these containments are strict
- What we can prove: At least one of these containments is strict. (Why?)

Complexity of CLIQUE

- Recall: $\mathrm{CLIQUE} = \{\langle G, k \rangle : G \text{ has a } k\text{-clique}\}$
- Last time, we discussed the fact that $\mathrm{CLIQUE} \in \mathrm{NP}$
- Consequence: If $\mathrm{P} = \mathrm{NP}$, then $\mathrm{CLIQUE} \in \mathrm{P}$
- **Plan for this week:** We will prove that if $\mathrm{P} \neq \mathrm{NP}$, then $\mathrm{CLIQUE} \notin \mathrm{P}$
  - This will provide evidence that $\mathrm{CLIQUE} \notin \mathrm{P}$
- To prove it, we will use concepts of NP-hardness and NP-completeness

NP-hardness

- **Definition:** Let L be a language. Suppose that for every $L' \in \mathrm{NP}$, there is a poly-time mapping reduction from L' to L. In this case, we say that L is NP-hard
- "L is NP-hard" means "L is at least as hard as every language in NP"

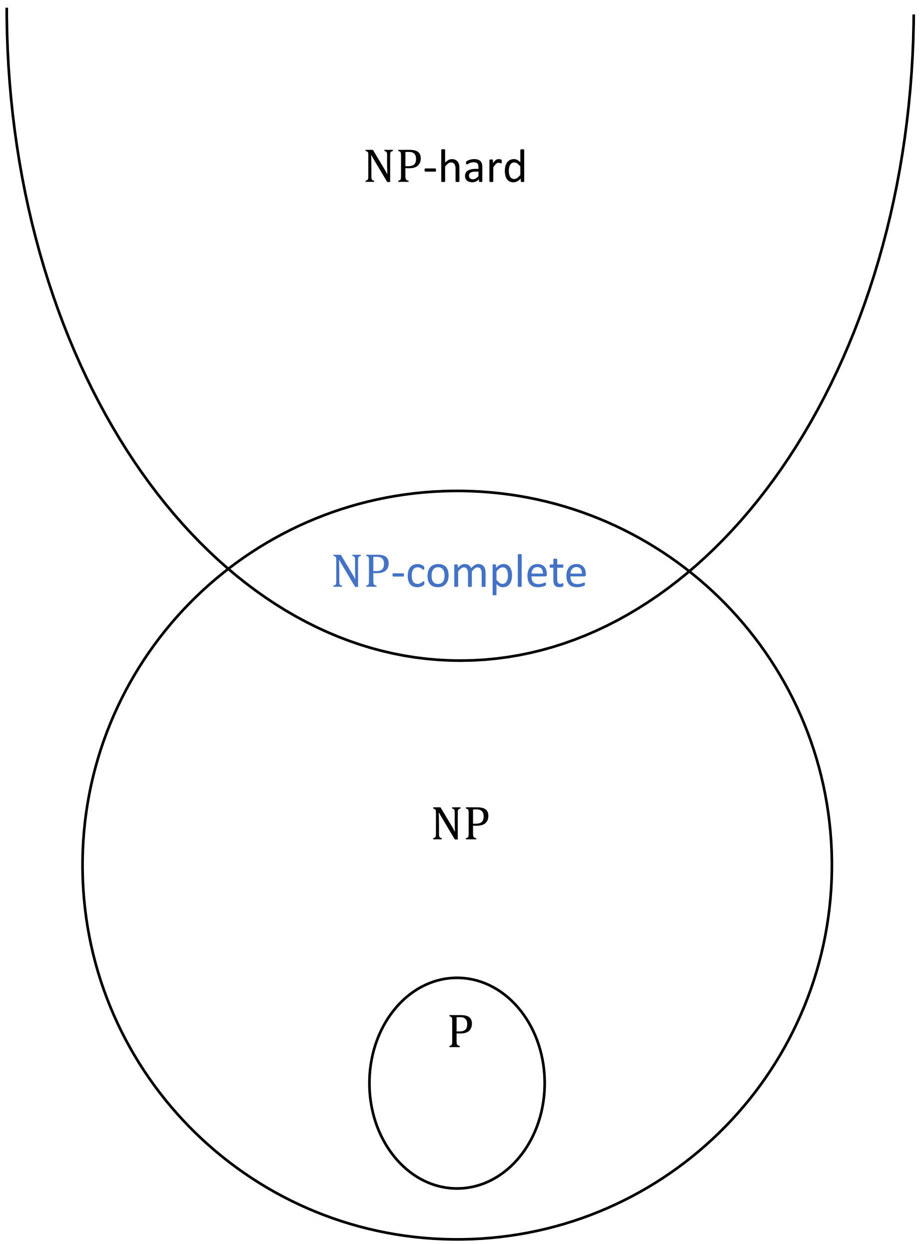
NP-completeness

- **Definition:** Let L be a language. We say that L is NP-complete if L is NP-hard and $L \in \mathrm{NP}$
- The NP-complete languages are the hardest languages in NP
- If L is NP-complete, then the language L can be said to "capture" / "express" the entire complexity class NP

NP-complete languages are probably not in P

- **Claim:** Suppose L is NP-complete. Then $L \in \mathrm{P}$ if and only if $\mathrm{P} = \mathrm{NP}$
- **Proof:** First, assume $\mathrm{P} = \mathrm{NP}$. Since $L \in \mathrm{NP}$, it follows that $L \in \mathrm{P}$ ✓
- Now assume $\mathrm{P} \neq \mathrm{NP}$, i.e., there is some language $L_{\mathrm{HARD}} \in \mathrm{NP} \setminus \mathrm{P}$
- By NP-hardness, there is a poly-time mapping reduction from L_{HARD} to L
- Since $L_{\mathrm{HARD}} \notin \mathrm{P}$, this implies $L \notin \mathrm{P}$ ✓

NP-completeness



Proving NP-completeness

- How can we prove that a language like CLIQUE is NP-complete?
- How can we use graph theory to simulate Turing machines?
- Key idea: Code as Data

Code as data, revisited

- Recall principle: A Turing machine M can be encoded as a string $\langle M \rangle$
  - M is an algorithm, but at the same time, $\langle M \rangle$ can be an input to another algorithm!
- Similar idea: A circuit C can be encoded as a string $\langle C \rangle$
  - You investigated ways to do this on problem set 5
  - C is an "algorithm," but at the same time, $\langle C \rangle$ can be an input to another algorithm!
  - What can we do with this idea?

Circuit value problem

- Let $\text{CIRCUIT-VALUE} = \{\langle C, x\rangle : C \text{ is a circuit and } C(x) = 1\}$
- **Claim:** $\text{CIRCUIT-VALUE} \in \text{P}$
- **Proof sketch:** Suppose C has m nodes. To compute $C(x)$:
  1) Mark all the input nodes with their values
  2) While there is an unmarked node:
     a) For every gate g, find all the nodes that feed into g. If they are all marked with their values, then mark g with its value

Constructing cir

Let x be an input to C. How should x be interpreted?

A: x is the description of a circuit that decides L

B: x is the description of a Turing machine that decides L

C: x is the binary encoding of a string in Σ^n

D: x is the binary encoding of a string in L

Respond at PollEv.com/whoza or text "whoza" to 22333

- Let $L \in \mathrm{P}$ where $L \subseteq \Sigma^*$
- Since $\mathrm{P} \subseteq \mathrm{PSIZE}$, we know

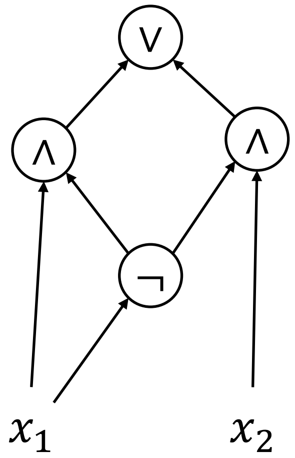
circuit that computes L_n, i.e., it decides L on inputs of length n

Theorem: There is a polynomial-time algorithm such that given 1^n, the algorithm outputs the description $\langle C \rangle$ of a circuit C that computes L_n

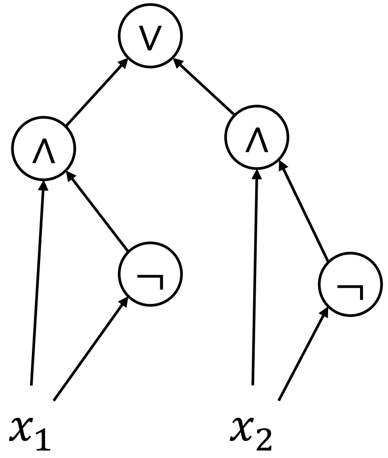
- **Proof sketch:** Use the circuit construction we used to prove $\mathrm{P} \subseteq \mathrm{PSIZE}$

Circuit satisfiability

- Let C be an n-input 1-output circuit
- We say that C is satisfiable if there exists $x \in \{0, 1\}^n$ such that $C(x) = 1$



Satisfiable



Unsatisfiable

Circuit satisfiability is NP-complete

- Let CIRCUIT-SAT $= \{\langle C \rangle : C$ is a satisfiable circuit$\}$

Theorem: CIRCUIT-SAT is NP-complete.

- Consequence: Studying CIRCUIT-SAT (one specific language) is equivalent to studying the abstract concept of "verifiability" (as modeled by the complexity class NP)

Proof that $\text{CIRCUIT-SAT} \in \text{NP}$

- Given $\langle C \rangle$, where C is an n-input 1-output circuit:
  1. Pick $x \in \{0, 1\}^n$ at random
  2. Check whether $C(x) = 1$ (recall $\text{CIRCUIT-VALUE} \in \text{P}$)
  3. Accept if $C(x) = 1$; reject if $C(x) = 0$

Proof that $\mathrm{CIRCUIT\text{-}SAT}$ is NP-hard

- Let L be any language in NP
- Our job: Design a mapping reduction from L to $\mathrm{CIRCUIT\text{-}SAT}$