Foundations of Computing

Lecture 23

Arkady Yerukhimovich

April 15, 2025

Remaining Class Schedule

- We have only 2 weeks left of lectures!
- HW7 is due tomorrow
- HW8 is out, due next Wednesday
- Thursday, April 24 will be a review lecture

Final Exam

Final exam will be on Tuesday, May 6, 10:20-12:20.

Outline

1. Lecture 22 Review
2. $\mathcal{NP}$-Intermediate Languages
3. co-$\mathcal{NP}$

Lecture 22 Review

- More $\mathcal{NP}$-complete problems
  - SAT
  - 3SAT
  - CLIQUE
  - VERTEX-COVER
  - NAE-SAT
  - 3-coloring

Vertex Cover Problem

Vertex Cover Problem

$$\text{VERTEX-COVER} = \{\langle G, k\rangle \mid G \text{ has a vertex cover of size } \leq k\}$$

Goal: Prove that VC is $\mathcal{NP}$-Complete

1. Show that VC $\in \mathcal{NP}$
2. Show that 3-SAT $\leq_p$ VC

3-SAT $\leq_p$ VC

Goal: Show reduction f from 3-SAT to VC s.t.

- if ϕ is satisfiable, $f(\phi) = \langle G, k \rangle$ s.t. G has VC of size $\leq k$
- if ϕ is not satisfiable, $f(\phi) = \langle G, k \rangle$ s.t. G has no VC of size $\leq k$

3-SAT $\leq_p$ VC

Goal: Show reduction f from 3-SAT to VC s.t.

- if ϕ is satisfiable, $f(\phi) = \langle G, k \rangle$ s.t. G has VC of size $\leq k$
- if ϕ is not satisfiable, $f(\phi) = \langle G, k \rangle$ s.t. G has no VC of size $\leq k$

Variable gadget: For every variable x_1, draw pair of nodes

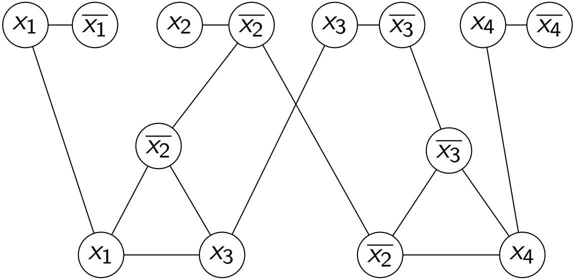
Clause gadget: For every (3-term) clause draw a triangle

Observations:

- For each variable need 1 node in cover
- For each triangle need at least 2 nodes
- Need to connect variables to clauses

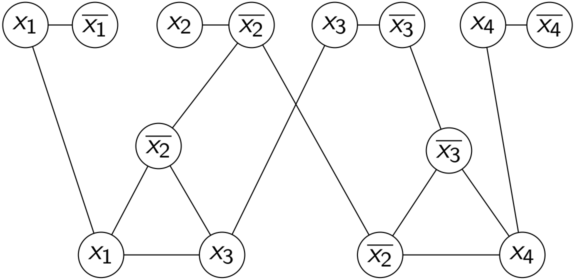
3-SAT $\leq_p$ VC Example

$$\phi = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_2} \vee \overline{x_3} \vee x_4)$$



3-SAT $\leq_p$ VC Example

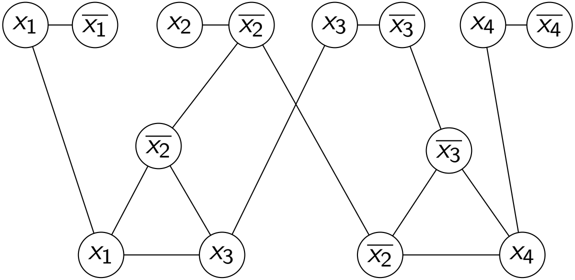
$$\phi = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_2} \vee \overline{x_3} \vee x_4)$$



1. A satisfying assignment implies cover C, $|C| \leq 2c + v$

3-SAT $\leq_p$ VC Example

$$\phi = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_2} \vee \overline{x_3} \vee x_4)$$



1. A satisfying assignment implies cover C, $|C| \leq 2c + v$
2. No satisfying assignment implies smallest cover needs $|C| > 2c + v$

Why This Works

Why This Works

Satisfying assignment $\Rightarrow |C| = 2c + v$:

Why This Works

Satisfying assignment $\Rightarrow |C| = 2c + v$:

- Include the node corresponding to the 1 value in C (i.e., if $x_1 = 1$ then include x_1, otherwise include $\overline{x_1}$).

Why This Works

Satisfying assignment $\Rightarrow |C| = 2c + v$:

- Include the node corresponding to the 1 value in C (i.e., if $x_1 = 1$ then include x_1, otherwise include $\overline{x_1}$).
- Since $\phi(x) = 1$, for every triangle at least one edge between triangle and variable gadgets is already covered (i.e., at least one variable in each clause is satisfied).

Why This Works

Satisfying assignment $\Rightarrow |C| = 2c + v$:

- Include the node corresponding to the 1 value in C (i.e., if $x_1 = 1$ then include x_1, otherwise include $\overline{x_1}$).
- Since $\phi(x) = 1$, for every triangle at least one edge between triangle and variable gadgets is already covered (i.e., at least one variable in each clause is satisfied).
- Can include the remaining two nodes in triangle in C to cover the remaining edges.

Why This Works

Satisfying assignment $\Rightarrow |C| = 2c + v$:

- Include the node corresponding to the 1 value in C (i.e., if $x_1 = 1$ then include x_1, otherwise include $\overline{x_1}$).
- Since $\phi(x) = 1$, for every triangle at least one edge between triangle and variable gadgets is already covered (i.e., at least one variable in each clause is satisfied).
- Can include the remaining two nodes in triangle in C to cover the remaining edges.
- This results in a cover of size $2c + v$.

Why This Works

Satisfying assignment $\Rightarrow |C| = 2c + v$:

- Include the node corresponding to the 1 value in C (i.e., if $x_1 = 1$ then include x_1, otherwise include $\overline{x_1}$).
- Since $\phi(x) = 1$, for every triangle at least one edge between triangle and variable gadgets is already covered (i.e., at least one variable in each clause is satisfied).
- Can include the remaining two nodes in triangle in C to cover the remaining edges.
- This results in a cover of size $2c + v$.

$|C| = 2c + v \Rightarrow$ Satisfying assignment

Why This Works

Satisfying assignment $\Rightarrow |C| = 2c + v$:

- Include the node corresponding to the 1 value in C (i.e., if $x_1 = 1$ then include x_1, otherwise include $\overline{x_1}$).
- Since $\phi(x) = 1$, for every triangle at least one edge between triangle and variable gadgets is already covered (i.e., at least one variable in each clause is satisfied).
- Can include the remaining two nodes in triangle in C to cover the remaining edges.
- This results in a cover of size $2c + v$.

$|C| = 2c + v \Rightarrow$ Satisfying assignment

- C must contain 2 nodes per triangle and 1 node per variable gadget

Why This Works

Satisfying assignment $\Rightarrow |C| = 2c + v$:

- Include the node corresponding to the 1 value in C (i.e., if $x_1 = 1$ then include x_1, otherwise include $\overline{x_1}$).
- Since $\phi(x) = 1$, for every triangle at least one edge between triangle and variable gadgets is already covered (i.e., at least one variable in each clause is satisfied).
- Can include the remaining two nodes in triangle in C to cover the remaining edges.
- This results in a cover of size $2c + v$.

$|C| = 2c + v \Rightarrow$ Satisfying assignment

- C must contain 2 nodes per triangle and 1 node per variable gadget
- At least one edge connecting each triangle to variable gadgets can not be covered by the triangle nodes (that would require all 3 nodes).

Why This Works

Satisfying assignment $\Rightarrow |C| = 2c + v$:

- Include the node corresponding to the 1 value in C (i.e., if $x_1 = 1$ then include x_1, otherwise include $\overline{x_1}$).
- Since $\phi(x) = 1$, for every triangle at least one edge between triangle and variable gadgets is already covered (i.e., at least one variable in each clause is satisfied).
- Can include the remaining two nodes in triangle in C to cover the remaining edges.
- This results in a cover of size $2c + v$.

$|C| = 2c + v \Rightarrow$ Satisfying assignment

- C must contain 2 nodes per triangle and 1 node per variable gadget
- At least one edge connecting each triangle to variable gadgets can not be covered by the triangle nodes (that would require all 3 nodes).
- This edge needs to be covered by the variable node.

Why This Works

Satisfying assignment $\Rightarrow |C| = 2c + v$:

- Include the node corresponding to the 1 value in C (i.e., if $x_1 = 1$ then include x_1, otherwise include $\overline{x_1}$).
- Since $\phi(x) = 1$, for every triangle at least one edge between triangle and variable gadgets is already covered (i.e., at least one variable in each clause is satisfied).
- Can include the remaining two nodes in triangle in C to cover the remaining edges.
- This results in a cover of size $2c + v$.

$|C| = 2c + v \Rightarrow$ Satisfying assignment

- C must contain 2 nodes per triangle and 1 node per variable gadget
- At least one edge connecting each triangle to variable gadgets can not be covered by the triangle nodes (that would require all 3 nodes).
- This edge needs to be covered by the variable node.
- Variable nodes in C must cover at least one edge to each triangle implying a satisfying assignment.

Outline

Ladner's Theorem

- Recall that we know that $\mathcal{P} \subseteq \mathcal{NP}$

Ladner's Theorem

- Recall that we know that $\mathcal{P} \subseteq \mathcal{NP}$
- Suppose that $\mathcal{P} \neq \mathcal{NP}$:

Ladner's Theorem

- Recall that we know that $\mathcal{P} \subseteq \mathcal{NP}$
- Suppose that $\mathcal{P} \neq \mathcal{NP}$:

  Question: Are all languages either easy or very hard?

Ladner's Theorem

- Recall that we know that $\mathcal{P} \subseteq \mathcal{NP}$
- Suppose that $\mathcal{P} \neq \mathcal{NP}$:

  Question: Are all languages either easy or very hard?

  Math version: Is there an $L \in \mathcal{NP}$, s.t. $L \notin \mathcal{P}$ and L is not $\mathcal{NP}$-Complete?

Ladner's Theorem

If $\mathcal{P} \neq \mathcal{NP}$ then there exists an $L \in \mathcal{NP}$ s.t.

1. $L \notin \mathcal{P}$, and
2. L is not $\mathcal{NP}$-Complete

Ladner's Theorem

- Recall that we know that $\mathcal{P} \subseteq \mathcal{NP}$
- Suppose that $\mathcal{P} \neq \mathcal{NP}$:

  Question: Are all languages either easy or very hard?

  Math version: Is there an $L \in \mathcal{NP}$, s.t. $L \notin \mathcal{P}$ and L is not $\mathcal{NP}$-Complete?

Ladner's Theorem

If $\mathcal{P} \neq \mathcal{NP}$ then there exists an $L \in \mathcal{NP}$ s.t.

1. $L \notin \mathcal{P}$, and
2. L is not $\mathcal{NP}$-Complete

Comment: All languages useful for crypto are such $\mathcal{NP}$-intermediate languages

Proof

A Useful Language

$$SAT_H = \{\phi 0 1^{n^{H(n)}} \mid \phi \in SAT, n = |\phi|\}$$

Proof

A Useful Language

$$SAT_H = \{\phi 01^{n^{H(n)}} \mid \phi \in SAT, n = |\phi|\}$$

1. If $H(n) = n$, then $SAT_H \in \mathcal{P}$

Proof

A Useful Language

$$SAT_H = \{\phi 0 1^{n^{H(n)}} \mid \phi \in SAT, n = |\phi|\}$$

1. If $H(n) = n$, then $SAT_H \in \mathcal{P}$
2. If $H(n) \leq c$, then SAT_H is $\mathcal{NP}$-Complete

Proof

A Useful Language

$$SAT_H = \{\phi 01^{n^{H(n)}} \mid \phi \in SAT, n = |\phi|\}$$

1. If $H(n) = n$, then $SAT_H \in \mathcal{P}$
2. If $H(n) \leq c$, then SAT_H is $\mathcal{NP}$-Complete
3. We will define H to be in between these two cases

Defining $H(n)$

Let $M_1, M_2, \ldots$ be an enumeration of all TM's (can do this since TM's are countable)

$H(n)$

- Smallest $i \leq \log\log n$ s.t. for all $x \in \{0,1\}^*$, with $|x| \leq \log n$

$$M_i(x) \text{ halts in } i|x|^i \text{ steps and accepts iff } x \in SAT_H$$

- If no such M_i exists, $H(n) = \log\log n$

Defining $H(n)$

Let $M_1, M_2, \ldots$ be an enumeration of all TM's (can do this since TM's are countable)

$H(n)$

- Smallest $i \leq \log\log n$ s.t. for all $x \in \{0,1\}^*$, with $|x| \leq \log n$

$$M_i(x) \text{ halts in } i|x|^i \text{ steps and accepts iff } x \in SAT_H$$

- If no such M_i exists, $H(n) = \log\log n$

1. $H(n)$ is computable since can enumerate all short x

Defining $H(n)$

Let $M_1, M_2, \ldots$ be an enumeration of all TM's (can do this since TM's are countable)

$H(n)$

- Smallest $i \leq \log\log n$ s.t. for all $x \in \{0,1\}^*$, with $|x| \leq \log n$

$$M_i(x) \text{ halts in } i|x|^i \text{ steps and accepts iff } x \in SAT_H$$

- If no such M_i exists, $H(n) = \log\log n$

1. $H(n)$ is computable since can enumerate all short x
2. Claim: $SAT_H \in \mathcal{P}$ iff $H(n) < c$ for all n

Defining $H(n)$

Let $M_1, M_2, \ldots$ be an enumeration of all TM's (can do this since TM's are countable)

$H(n)$

- Smallest $i \leq \log \log n$ s.t. for all $x \in \{0,1\}^*$, with $|x| \leq \log n$

$$M_i(x) \text{ halts in } i|x|^i \text{ steps and accepts iff } x \in SAT_H$$

- If no such M_i exists, $H(n) = \log \log n$

1. $H(n)$ is computable since can enumerate all short x
2. Claim: $SAT_H \in \mathcal{P}$ iff $H(n) < c$ for all n
   ($\Rightarrow$) By definition of $\mathcal{P}$, there is machine M_k that decides SAT_H in kn^k steps so $H(n) = k$

Claim

$SAT_H \in \mathcal{P}$ iff $H(n) < c$ for all n

- Already proved ($\Rightarrow$)

Claim

$SAT_H \in \mathcal{P}$ iff $H(n) < c$ for all n

- Already proved ($\Rightarrow$)
- ($\Leftarrow$) If $H(n) < c$ for all $n \to \infty$, then there is infinitely long stretch where $H(n) = i$ for some $1 \leq i \leq c$.

Claim

$SAT_H \in \mathcal{P}$ iff $H(n) < c$ for all n

- Already proved ($\Rightarrow$)
- ($\Leftarrow$) If $H(n) < c$ for all $n \to \infty$, then there is infinitely long stretch where $H(n) = i$ for some $1 \leq i \leq c$.
  - We will prove that in this case M_i decides SAT_H

Claim

$SAT_H \in \mathcal{P}$ iff $H(n) < c$ for all n

- Already proved ($\Rightarrow$)
- ($\Leftarrow$) If $H(n) < c$ for all $n \to \infty$, then there is infinitely long stretch where $H(n) = i$ for some $1 \leq i \leq c$.
  - We will prove that in this case M_i decides SAT_H
  - Assume that M_i does not halt on some x after $i|x|^i$ steps

Claim

$SAT_H \in \mathcal{P}$ iff $H(n) < c$ for all n

- Already proved $(\Rightarrow)$
- $(\Leftarrow)$ If $H(n) < c$ for all $n \to \infty$, then there is infinitely long stretch where $H(n) = i$ for some $1 \leq i \leq c$.
  - We will prove that in this case M_i decides SAT_H
  - Assume that M_i does not halt on some x after $i|x|^i$ steps
  - Let $n > 2^{|x|}$. For any such n, if $H(n) = i$, $M_i(x)$ must halt in $i|x|^i$ steps.

Claim

$SAT_H \in \mathcal{P}$ iff $H(n) < c$ for all n

- Already proved ($\Rightarrow$)
- ($\Leftarrow$) If $H(n) < c$ for all $n \to \infty$, then there is infinitely long stretch where $H(n) = i$ for some $1 \leq i \leq c$.
  - We will prove that in this case M_i decides SAT_H
  - Assume that M_i does not halt on some x after $i|x|^i$ steps
  - Let $n > 2^{|x|}$. For any such n, if $H(n) = i$, $M_i(x)$ must halt in $i|x|^i$ steps.
  - So, $H(n) \neq i$ for all $n > 2^{|x|}$. Contradiction!

Completing the proof

Claim

$SAT_H \in \mathcal{P}$ iff $H(n) < c$ for all n

Completing the proof:

Completing the proof

Claim

$SAT_H \in \mathcal{P}$ iff $H(n) < c$ for all n

Completing the proof:

1. $SAT_H \notin \mathcal{P}$:
    - Suppose it is in $\mathcal{P}$, then $H(n) < c$
    - Can reduce any SAT formula to SAT_H formula by padding with $H(n)$ 1s
    - But, SAT is $\mathcal{NP}$-Complete, contradiction!

Completing the proof

Claim

$SAT_H \in \mathcal{P}$ iff $H(n) < c$ for all n

Completing the proof:

1. $SAT_H \notin \mathcal{P}$:
   - Suppose it is in $\mathcal{P}$, then $H(n) < c$
   - Can reduce any SAT formula to SAT_H formula by padding with $H(n)$ 1s
   - But, SAT is $\mathcal{NP}$-Complete, contradiction!
2. SAT_H is not $\mathcal{NP}$-Complete

Completing the proof

Claim

$SAT_H \in \mathcal{P}$ iff $H(n) < c$ for all n

Completing the proof:

1. $SAT_H \notin \mathcal{P}$:
   - Suppose it is in $\mathcal{P}$, then $H(n) < c$
   - Can reduce any SAT formula to SAT_H formula by padding with $H(n)$ 1s
   - But, SAT is $\mathcal{NP}$-Complete, contradiction!
2. SAT_H is not $\mathcal{NP}$-Complete
   - Assume it is, then $SAT \leq_p SAT_H$

Completing the proof

Claim

$SAT_H \in \mathcal{P}$ iff $H(n) < c$ for all n

Completing the proof:

1. $SAT_H \notin \mathcal{P}$:
   - Suppose it is in $\mathcal{P}$, then $H(n) < c$
   - Can reduce any SAT formula to SAT_H formula by padding with $H(n)$ 1s
   - But, SAT is $\mathcal{NP}$-Complete, contradiction!
2. SAT_H is not $\mathcal{NP}$-Complete
   - Assume it is, then $SAT \leq_p SAT_H$
   - Reduction maps ψ of length n to $\phi 01^{H(n)}$ where $|\phi 01^{H(n)}| = n^c$

Completing the proof

Claim

$SAT_H \in \mathcal{P}$ iff $H(n) < c$ for all n

Completing the proof:

1. $SAT_H \notin \mathcal{P}$:
   - Suppose it is in $\mathcal{P}$, then $H(n) < c$
   - Can reduce any SAT formula to SAT_H formula by padding with $H(n)$ 1s
   - But, SAT is $\mathcal{NP}$-Complete, contradiction!
2. SAT_H is not $\mathcal{NP}$-Complete
   - Assume it is, then $SAT \leq_p SAT_H$
   - Reduction maps ψ of length n to $\phi 01^{H(n)}$ where $|\phi 01^{H(n)}| = n^c$
   - But $H(n) \to \infty$ so the padding is super-poly in size of ϕ

Completing the proof

Claim

$SAT_H \in \mathcal{P}$ iff $H(n) < c$ for all n

Completing the proof:

1. $SAT_H \notin \mathcal{P}$:
   - Suppose it is in $\mathcal{P}$, then $H(n) < c$
   - Can reduce any SAT formula to SAT_H formula by padding with $H(n)$ 1s
   - But, SAT is $\mathcal{NP}$-Complete, contradiction!
2. SAT_H is not $\mathcal{NP}$-Complete
   - Assume it is, then $SAT \leq_p SAT_H$
   - Reduction maps ψ of length n to $\phi 01^{H(n)}$ where $|\phi 01^{H(n)}| = n^c$
   - But $H(n) \to \infty$ so the padding is super-poly in size of ϕ
   - Hence $|\phi| << n$, so have reduced solving long formula to solving a much shorter one.

Completing the proof

Claim

$SAT_H \in \mathcal{P}$ iff $H(n) < c$ for all n

Completing the proof:

1. $SAT_H \notin \mathcal{P}$:
   - Suppose it is in $\mathcal{P}$, then $H(n) < c$
   - Can reduce any SAT formula to SAT_H formula by padding with $H(n)$ 1s
   - But, SAT is $\mathcal{NP}$-Complete, contradiction!
2. SAT_H is not $\mathcal{NP}$-Complete
   - Assume it is, then $SAT \leq_p SAT_H$
   - Reduction maps ψ of length n to $\phi 01^{H(n)}$ where $|\phi 01^{H(n)}| = n^c$
   - But $H(n) \to \infty$ so the padding is super-poly in size of ϕ
   - Hence $|\phi| << n$, so have reduced solving long formula to solving a much shorter one.
   - Repeat this enough times to make $|\phi| = O(1)$ and solve.

Takeaway

If $\mathcal{P} \neq \mathcal{NP}$, then $\mathcal{NP}$-intermediate languages exist!

Outline

Are All Problems in $\mathcal{NP}$?

Question

Do all languages have poly-size witnesses?

Are All Problems in $\mathcal{NP}$?

Question

Do all languages have poly-size witnesses?

Consider the following language:

UNSAT

$$\text{UNSAT} = \{\langle \phi \rangle \mid \phi \text{ is not satisfiable}\}$$

Are All Problems in $\mathcal{NP}$?

Question

Do all languages have poly-size witnesses?

Consider the following language:

UNSAT

$$\mathsf{UNSAT} = \{\langle\phi\rangle \mid \phi \text{ is not satisfiable}\}$$

- For all possible assignments $w \in \{0,1\}^n$, $\phi(w) = 0$

Are All Problems in $\mathcal{NP}$?

Question

Do all languages have poly-size witnesses?

Consider the following language:

UNSAT

$$\text{UNSAT} = \{\langle\phi\rangle \mid \phi \text{ is not satisfiable}\}$$

- For all possible assignments $w \in \{0,1\}^n$, $\phi(w) = 0$
- Note that this language consists of all formulas that are not in SAT. I.e., UNSAT$=\overline{SAT}$

Are All Problems in $\mathcal{NP}$?

Question

Do all languages have poly-size witnesses?

Consider the following language:

UNSAT

$$\text{UNSAT} = \{\langle\phi\rangle \mid \phi \text{ is not satisfiable}\}$$

- For all possible assignments $w \in \{0,1\}^n$, $\phi(w) = 0$
- Note that this language consists of all formulas that are not in SAT. I.e., UNSAT$=\overline{SAT}$

Question

Is UNSAT in $\mathcal{NP}$?

co-$\mathcal{NP}$

We define a new complexity class co-$\mathcal{NP}$ to capture such languages.

Co-NP is the class of languages that:

We define a new complexity class co-$\mathcal{NP}$ to capture such languages.

Co-NP is the class of languages that:

1. $\{L \mid \overline{L} \in \mathcal{NP}\}$

co-$\mathcal{NP}$

We define a new complexity class co-$\mathcal{NP}$ to capture such languages.

Co-NP is the class of languages that:

1. $\{L \mid \overline{L} \in \mathcal{NP}\}$
2. There exists poly-time DTM V s.t. for $x \in L$ <u>for all</u> w, $V(x, w) = 0$

Observations:

- co-$\mathcal{NP}$ is not the complement of $\mathcal{NP}$ (i.e., it does not consist of all languages not in $\mathcal{NP}$)

co-$\mathcal{NP}$

We define a new complexity class co-$\mathcal{NP}$ to capture such languages.

Co-NP is the class of languages that:

1. $\{L \mid \overline{L} \in \mathcal{NP}\}$
2. There exists poly-time DTM V s.t. for $x \in L$ for all w, $V(x, w) = 0$

Observations:

- co-$\mathcal{NP}$ is not the complement of $\mathcal{NP}$ (i.e., it does not consist of all languages not in $\mathcal{NP}$)
- In particular, there are many languages in $\mathcal{NP} \cap$ co-$\mathcal{NP}$

co-$\mathcal{NP}$

We define a new complexity class co-$\mathcal{NP}$ to capture such languages.

Co-NP is the class of languages that:

1. $\{L \mid \overline{L} \in \mathcal{NP}\}$
2. There exists poly-time DTM V s.t. for $x \in L$ <u>for all</u> w, $V(x, w) = 0$

Observations:

- co-$\mathcal{NP}$ is not the complement of $\mathcal{NP}$ (i.e., it does not consist of all languages not in $\mathcal{NP}$)
- In particular, there are many languages in $\mathcal{NP} \cap$ co-$\mathcal{NP}$
- In fact, $\mathcal{P} \subseteq (\mathcal{NP} \cap$ co-$\mathcal{NP})$

$\mathcal{P}$, $\mathcal{NP}$ and co-$\mathcal{NP}$

$\mathcal{P}$

$L \in \mathcal{P}$ if there exists poly-time DTM M s.t $M(x) = [x \in L]$

$\mathcal{P}$, $\mathcal{NP}$ and co-$\mathcal{NP}$

$\mathcal{P}$

$L \in \mathcal{P}$ if there exists poly-time DTM M s.t $M(x) = [x \in L]$

$\mathcal{NP}$

$L \in \mathcal{NP}$ if there exists poly-time DTM V s.t. for $x \in L$ there exists a witness w s.t. $V(x, w) = 1$

$\mathcal{P}$, $\mathcal{NP}$ and co-$\mathcal{NP}$

$\mathcal{P}$

$L \in \mathcal{P}$ if there exists poly-time DTM M s.t $M(x) = [x \in L]$

$\mathcal{NP}$

$L \in \mathcal{NP}$ if there exists poly-time DTM V s.t. for $x \in L$ there exists a witness w s.t. $V(x, w) = 1$

co-$\mathcal{NP}$

$L \in$ co-$\mathcal{NP}$ if there exists poly-time DTM V s.t. for $x \in L$ for all w, $V(x, w) = 0$

$\mathcal{P}$, $\mathcal{NP}$ and co-$\mathcal{NP}$

$\mathcal{P}$

$L \in \mathcal{P}$ if there exists poly-time DTM M s.t $M(x) = [x \in L]$

$\mathcal{NP}$

$L \in \mathcal{NP}$ if there exists poly-time DTM V s.t. for $x \in L$ there exists a witness w s.t. $V(x, w) = 1$

co-$\mathcal{NP}$

$L \in$ co-$\mathcal{NP}$ if there exists poly-time DTM V s.t. for $x \in L$ for all w, $V(x, w) = 0$

Question:

Can you prove that $x \in L$, when $L \in$co-$\mathcal{NP}$?

Proving that $x \in L$ for $L \in$co-$\mathcal{NP}$

The Problem

Suppose, I am given an input formula ϕ and I want to prove that ϕ is not satisfiable.

Proving that $x \in L$ for $L \in$co-$\mathcal{NP}$

The Problem

Suppose, I am given an input formula ϕ and I want to prove that ϕ is not satisfiable.

- It is widely believed that there is no poly-size, efficiently verifiable proof w that you could give for UNSAT

Proving that $x \in L$ for $L \in$co-$\mathcal{NP}$

The Problem

Suppose, I am given an input formula ϕ and I want to prove that ϕ is not satisfiable.

- It is widely believed that there is no poly-size, efficiently verifiable proof w that you could give for UNSAT
- I.e., $\mathcal{NP} \neq$ co-$\mathcal{NP}$

Proving that $x \in L$ for $L \in$co-$\mathcal{NP}$

The Problem

Suppose, I am given an input formula ϕ and I want to prove that ϕ is not satisfiable.

- It is widely believed that there is no poly-size, efficiently verifiable proof w that you could give for UNSAT
- I.e., $\mathcal{NP} \neq$ co-$\mathcal{NP}$
- But, we don't know how to prove this

The Complexity Zoo

- There are many other complexity classes

The Complexity Zoo

- There are many other complexity classes
- We know some relationships between classes

The Complexity Zoo

- There are many other complexity classes
- We know some relationships between classes
- But, most big questions (e.g., $\mathcal{P} = \mathcal{NP}$, $\mathcal{NP} =$ co-$\mathcal{NP}$, etc.) are still not known!!!

The Complexity Zoo

- There are many other complexity classes
- We know some relationships between classes
- But, most big questions (e.g., $\mathcal{P} = \mathcal{NP}$, $\mathcal{NP} =$ co-$\mathcal{NP}$, etc.) are still not known!!!

Complexity Zoo

The complexity zoo (`https://complexityzoo.net/Complexity_Zoo`) now has 550 complexity classes.