Computational Complexity Theory

Lecture 6: NTM; Class co-NP and EXP

Department of Computer Science,
Indian Institute of Science

Recap: Search version of NP problems

- Recall: A language $L \subseteq \{0,1\}^*$ is in NP if
  - There's a *poly-time verifier* M and *poly. function* p s.t.
  - $x \in L$ iff there's a $u \in \{0,1\}^{p(|x|)}$ s.t $M(x, u) = 1$.

- Search version of L: Given an input $x \in \{0,1\}^*$, *find* a $u \in \{0,1\}^{p(|x|)}$ such that $M(x, u) = 1$, if such a u exists.

- Remark: Search version of L only makes sense once we have a verifier M in mind.

Recap: Decision versus Search

- Is the search version of an NP problem more difficult than the corresponding decision version?

- Theorem. Let $L \subseteq \{0,1\}^*$ be NP-complete. Then, the search version of L can be solved in poly-time if and only if the decision version can be solved in poly-time.

w.r.t any verifier M !

Recap: Decision versus Search

- Is *search* equivalent to *decision* for every NP problem?

- Theorem. *(Bellare & Goldwasser 1994)* If EE $\neq$ NEE then there's a language in NP for which search does not reduce to decision.

- Sometimes, the decision version of a problem can be trivial but the search version is possibly hard. E.g., Computing Nash Equilibrium (see class PPAD).

Recap: Two types of poly-time reductions

- Definition. A language $L_1 \subseteq \{0,1\}^*$ is *polynomial-time (Karp or many-one) reducible* to a language $L_2 \subseteq \{0,1\}^*$ if there's a polynomial time computable function f s.t.

$$x \in L_1 \Leftrightarrow f(x) \in L_2$$

- Definition. A language $L_1 \subseteq \{0,1\}^*$ is *polynomial-time (Cook or Turing) reducible* to a language $L_2 \subseteq \{0,1\}^*$ if there's a TM that decides L_1 in poly-time using poly-many calls to a "subroutine" (*oracle*) for deciding L_2.

Karp reduction implies Cook reduction

NTM: An alternate characterization of NP

Nondeterministic Turing Machines

- A *nondeterministic Turing machine* is like a deterministic Turing machines but with two transition functions.
- It is formally defined by a tuple $(\Gamma, Q, \delta_0, \delta_1)$. It has a special state q_{accept} in addition to q_{start} and q_{halt}.

Nondeterministic Turing Machines

- A *nondeterministic Turing machine* is like a deterministic Turing machines but with two transition functions.
- It is formally defined by a tuple $(\Gamma, Q, \delta_0, \delta_1)$. It has a special state q_{accept} in addition to q_{start} and q_{halt}.
- At every step of computation, the machine applies one of two functions δ_0 and δ_1 <u>*arbitrarily*</u>.

also called <u>*nondeterministically*</u>

Nondeterministic Turing Machines

- A *nondeterministic Turing machine* is like a deterministic Turing machines but with two transition functions.
- It is formally defined by a tuple $(\Gamma, Q, \delta_0, \delta_1)$. It has a special state q_{accept} in addition to q_{start} and q_{halt}.
- At every step of computation, the machine applies one of two functions δ_0 and δ_1 *arbitrarily*.

this is different from *randomly*

Nondeterministic Turing Machines

- A *nondeterministic Turing machine* is like a deterministic Turing machines but with two transition functions.
- It is formally defined by a tuple $(\Gamma, Q, \delta_0, \delta_1)$. It has a special state q_{accept} in addition to q_{start} and q_{halt}.
- At every step of computation, the machine applies one of two functions δ_0 and δ_1 *arbitrarily*.
- Unlike DTMs, NTMs are **not intended to be physically realizable** (because of the arbitrary nature of application of the transition functions).

Nondeterministic Turing Machines

- Definition. An NTM M *accepts* a string $x \in \{0,1\}^*$ iff on input x there ***exists*** a sequence of applications of the transition functions δ_0 and δ_1 (beginning from the start configuration) that makes M reach q_{accept}.

- Defintion. An NTM M *decides* a language $L \subseteq \{0,1\}^*$ if
  - M accepts x $\Longleftrightarrow$ $x \in L$
  - On every sequence of applications of the transition functions on input x, M either reaches q_{accept} or q_{halt}.

Nondeterministic Turing Machines

- Definition. An NTM M *accepts* a string $x \in \{0,1\}^*$ iff on input x there ***exists*** a sequence of applications of the transition functions δ_0 and δ_1 (beginning from the start configuration) that makes M reach q_{accept}.

- Defintion. An NTM M *decides* a language $L \subseteq \{0,1\}^*$ if
  - M accepts $x \Longleftrightarrow x \in L$
  - On every sequence of applications of the transition functions on input x, M either reaches q_{accept} or q_{halt}.

remember in this course we'll always be dealing with TMs that halt on every input.

Nondeterministic Turing Machines

- Definition. An NTM M *accepts* a string $x \in \{0,1\}^*$ iff on input x there ***exists*** a sequence of applications of the transition functions δ_0 and δ_1 (beginning from the start configuration) that makes M reach q_{accept}.

- Defintion. An NTM M *decides* L in $T(|x|)$ time if
  - M accepts x $\Longleftrightarrow$ $x \in L$
  - On <u>every sequence</u> of applications of the transition functions on input x, M either reaches q_{accept} or q_{halt} within $T(|x|)$ steps of computation.

Class NTIME

- Definition. A language L is in $NTIME(T(n))$ if there's an NTM M that decides L in $c.\ T(n)$ time on inputs of length n, where c is a constant.

Alternate characterization of NP

- Definition. A language L is in $\mathrm{NTIME}(T(n))$ if there's an NTM M that decides L in $c.\,T(n)$ time on inputs of length n, where c is a constant.

- Theorem. $\mathrm{NP} = \bigcup_{c>0} \mathrm{NTIME}(n^c)$.

  Proof sketch: Let L be a language in NP. Then, there's a poly-time verifier M s.t,

$$x \in L \iff \exists u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x,u) = 1$$

Alternate characterization of NP

- Definition. A language L is in $\mathrm{NTIME}(T(n))$ if there's an NTM M that decides L in $c.\,T(n)$ time on inputs of length n, where c is a constant.

- Theorem. $\mathrm{NP} = \bigcup_{c>0} \mathrm{NTIME}(n^c)$.

  Proof sketch: Let L be a language in NP. Then, there's a poly-time verifier M s.t,

$$x \in L \iff \exists u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x,u) = 1$$

Think of an NTM M' that on input x, at first <u>*guesses*</u> a $u \in \{0,1\}^{p(|x|)}$ by applying δ_0 and δ_1 nondeterministically

Alternate characterization of NP

- Definition. A language L is in $NTIME(T(n))$ if there's an NTM M that decides L in $c.\, T(n)$ time on inputs of length n, where c is a constant.

- Theorem. $NP = \bigcup_{c>0} NTIME\,(n^c)$.

  Proof sketch: Let L be a language in NP. Then, there's a poly-time verifier M s.t,

$$x \in L \iff \exists u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x, u) = 1$$

.... and then simulates M on (x, u) to verify $M(x,u) = 1$.

Alternate characterization of NP

- Definition. A language L is in $NTIME(T(n))$ if there's an NTM M that decides L in $c.\ T(n)$ time on inputs of length n, where c is a constant.

- Theorem. $NP = \bigcup_{c>0} NTIME(n^c)$.

  Proof sketch: Let L be in $NTIME(n^c)$. Then, there's an NTM M' that decides L in $p(n) = O(n^c)$ time. $(|x| = n)$

Alternate characterization of NP

- Definition. A language L is in NTIME(T(n)) if there's an NTM M that decides L in c. T(n) time on inputs of length n, where c is a constant.

- Theorem. $NP = \bigcup_{c > 0} NTIME(n^c)$.

  Proof sketch: Let L be in NTIME (n^c). Then, there's an NTM M' that decides L in $p(n) = O(n^c)$ time. ($|x| = n$)

Think of a verifier M that takes x and $u \in \{0,1\}^{p(n)}$ as input,

Alternate characterization of NP

- Definition. A language L is in $NTIME(T(n))$ if there's an NTM M that decides L in $c.\ T(n)$ time on inputs of length n, where c is a constant.

- Theorem. $NP = \bigcup_{c>0} NTIME(n^c)$.

Proof sketch: Let L be in $NTIME(n^c)$. Then, there's an NTM M' that decides L in $p(n) = O(n^c)$ time. ($|x| = n$)

Think of a verifier M that takes x and $u \in \{0,1\}^{p(n)}$ as input, and simulates M' on x with u as the sequence of choices for applying δ_0 and δ_1 .

Class co-NP

- Definition. For every $L \subseteq \{0,1\}^*$ let $\overline{L} = \{0,1\}^* \setminus L$.
  A language L is in co-NP if $\overline{L}$ is in NP.

- Example. $\overline{\mathrm{SAT}} = \{\phi : \phi$ is *not* satisfiable$\}$.

Class co-NP

- Definition. For every $L \subseteq \{0,1\}^*$ let $\overline{L} = \{0,1\}^* \setminus L$.
  A language L is in co-NP if $\overline{L}$ is in NP.

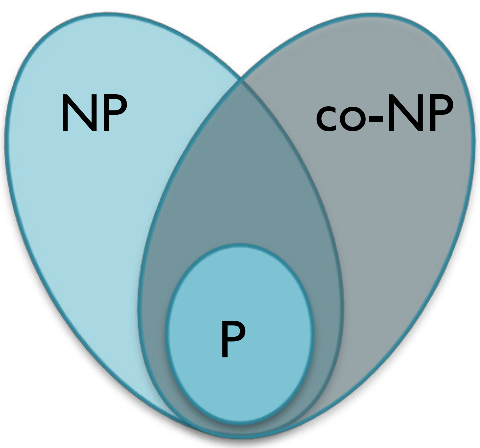
- Example. $\overline{\mathrm{SAT}} = \{\phi : \phi$ is *not* satisfiable$\}$.

- Note: co-NP is ***not*** complement of NP. Every language in P is in both NP and co-NP.

Class co-NP

- **Definition.** For every $L \subseteq \{0,1\}^*$ let $\overline{L} = \{0,1\}^* \setminus L$.
  A language L is in co-NP if $\overline{L}$ is in NP.

- **Example.** $\overline{\text{SAT}} = \{\phi : \phi$ is *not* satisfiable$\}$.



Class co-NP

- Definition. For every $L \subseteq \{0,1\}^*$ let $\overline{L} = \{0,1\}^* \setminus L$. A language L is in co-NP if $\overline{L}$ is in NP.

- Example. $\overline{\text{SAT}} = \{\phi : \phi \text{ is } \underline{\textit{not}} \text{ satisfiable}\}$.

- Note: $\overline{\text{SAT}}$ is Cook reducible to SAT. But, there's a fundamental difference between the two problems that is captured by the fact that $\overline{\text{SAT}}$ is <u>not</u> known to be Karp reducible to SAT. In other words, there's no known poly-time verification process for $\overline{\text{SAT}}$.

Class co-NP : Alternate definition

- Recall, a language $L \subseteq \{0,1\}^*$ is in NP if there's a *poly-time verifier* M such that

$$x \in L \Leftrightarrow \exists u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x, u) = 1$$

Class co-NP : Alternate definition

- Recall, a language $L \subseteq \{0,1\}^*$ is in NP if there's a *poly-time verifier* M such that

$$x \in L \iff \exists u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x, u) = 1$$

$$x \in \overline{L} \iff \forall u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x, u) = 0$$

Class co-NP : Alternate definition

- Recall, a language $L \subseteq \{0,1\}^*$ is in NP if there's a *poly-time verifier* M such that

$$x \in L \iff \exists u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x,u) = 1$$

$$x \in \overline{L} \iff \forall u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x,u) = 0$$

$$x \in \overline{L} \iff \forall u \in \{0,1\}^{p(|x|)} \text{ s.t. } \overline{M}(x,u) = 1$$

$\overline{M}$ outputs the *opposite* of M

Class co-NP : Alternate definition

- Recall, a language $L \subseteq \{0,1\}^*$ is in NP if there's a *poly-time verifier* M such that

$$x \in L \iff \exists u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x, u) = 1$$

$$x \in \overline{L} \iff \forall u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x, u) = 0$$

$$x \in \overline{L} \iff \forall u \in \{0,1\}^{p(|x|)} \text{ s.t. } \overline{M}(x, u) = 1$$

$\overline{M}$ is a poly-time TM

Class co-NP : Alternate definition

- Recall, a language $L \subseteq \{0,1\}^*$ is in NP if there's a *poly-time verifier* M such that

$$x \in L \iff \exists u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x, u) = 1$$

$$x \in \overline{L} \iff \forall u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x, u) = 0$$

$$x \in \overline{L} \iff \forall u \in \{0,1\}^{p(|x|)} \text{ s.t. } \overline{M}(x, u) = 1$$

is in co-NP

Class co-NP : Alternate definition

- Recall, a language $L \subseteq \{0,1\}^*$ is in NP if there's a *poly-time verifier* M such that

$$x \in L \iff \exists u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x, u) = 1$$

$$x \in \bar{L} \iff \forall u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x, u) = 0$$

$$x \in \bar{L} \iff \forall u \in \{0,1\}^{p(|x|)} \text{ s.t. } \bar{M}(x, u) = 1$$

- Definition. A language $L \subseteq \{0,1\}^*$ is in co-NP if there's a polynomial function p and a *poly-time TM* M such that

$$x \in L \iff \forall u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x, u) = 1$$

for NP this was $\exists$

co-NP-completeness

- Definition. A language $L' \subseteq \{0,1\}^*$ is co-NP-complete if
  - L' is in co-NP
  - Every language L in co-NP is polynomial-time (Karp) reducible to L'.

- Theorem. $\overline{SAT}$ is co-NP-complete.

co-NP-completeness

- Definition. A language $L' \subseteq \{0,1\}^*$ is co-NP-complete if
  - L' is in co-NP
  - Every language L in co-NP is polynomial-time (Karp) reducible to L'.

- Theorem. $\overline{\text{SAT}}$ is co-NP-complete.

Proof. Let $L \in$ co-NP. Then

$\overline{L} \in$ NP

co-NP-completeness

- Definition. A language $L' \subseteq \{0,1\}^*$ is co-NP-complete if
  - L' is in co-NP
  - Every language L in co-NP is polynomial-time (Karp) reducible to L'.

- Theorem. $\overline{SAT}$ is co-NP-complete.

Proof. Let $L \in$ co-NP. Then

$$\overline{L} \in NP$$

$$\Longrightarrow \overline{L} \leq_p SAT$$

co-NP-completeness

- Definition. A language $L' \subseteq \{0,1\}^*$ is co-NP-complete if
  - L' is in co-NP
  - Every language L in co-NP is polynomial-time (Karp) reducible to L'.

- Theorem. $\overline{SAT}$ is co-NP-complete.

  Proof. Let $L \in$ co-NP. Then

$$\overline{L} \in NP$$

$$\Rightarrow \overline{L} \leq_p SAT$$

$$\Rightarrow L \leq_p \overline{SAT}$$

co-NP-completeness

- Definition. A language $L' \subseteq \{0,1\}^*$ is co-NP-complete if
  - L' is in co-NP
  - Every language L in co-NP is polynomial-time (Karp) reducible to L'.

- Theorem. Let

$$\text{TAUTOLOGY} = \{\phi : \text{ every assignment satisfies } \phi\}.$$

TAUTOLOGY is co-NP-complete.

Proof. Similar (homework)

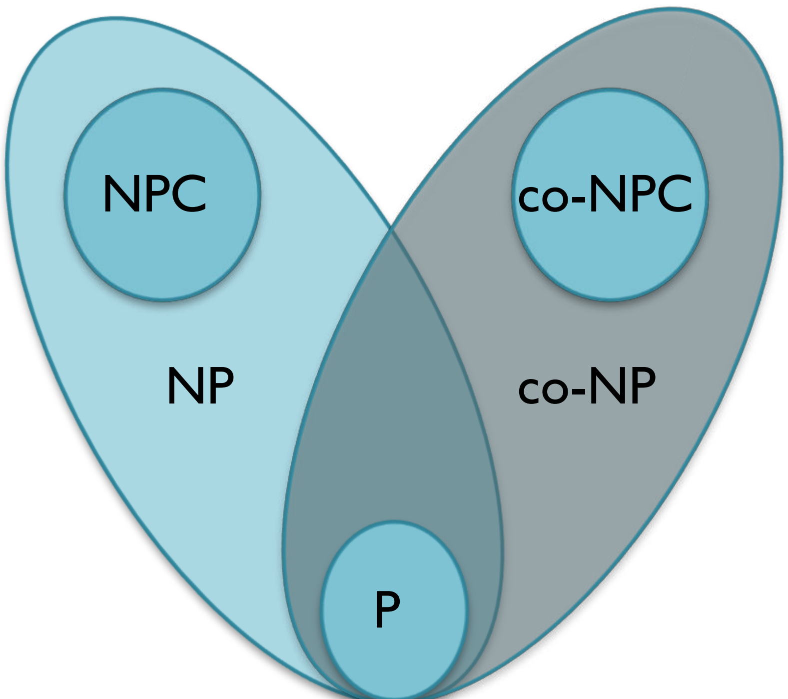
co-NP-completeness

- Definition. A language $L' \subseteq \{0,1\}^*$ is co-NP-complete if
  - L' is in co-NP
  - Every language L in co-NP is polynomial-time (Karp) reducible to L'.

- Theorem. If L in NP-complete then $\overline{L}$ is co-NP-complete

  Proof. Similar (homework)

The diagram again



If a co-NP-complete language belongs to NP then

co-NP $\subseteq$ NP

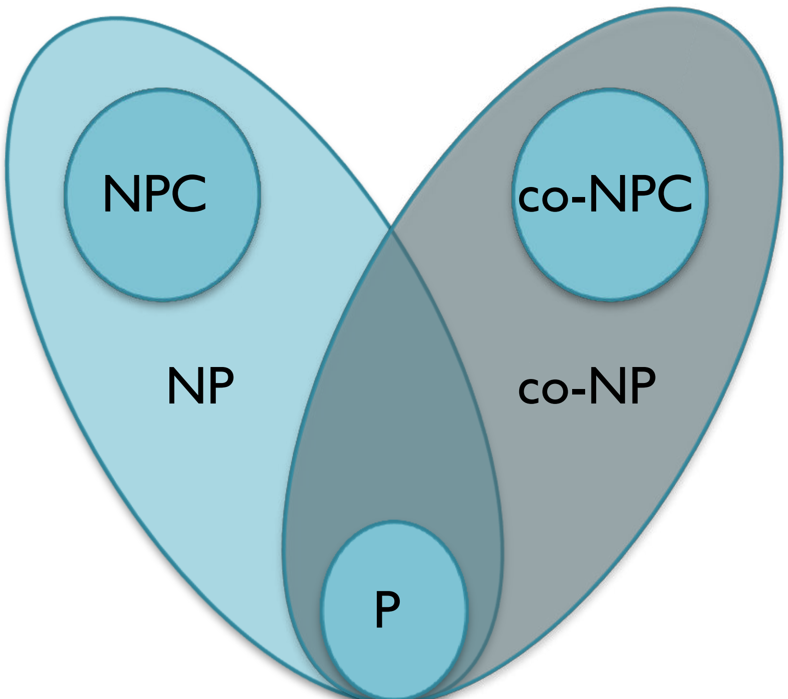
$\Rightarrow$ co-NP = NP

Let C_1 and C_2 be two complexity classes.

If $C_1 \subseteq C_2$, then co-$C_1 \subseteq$ co-C_2.

Obs. co-(co-C) = C .

The diagram again



If an NP-complete language belongs to co-NP then

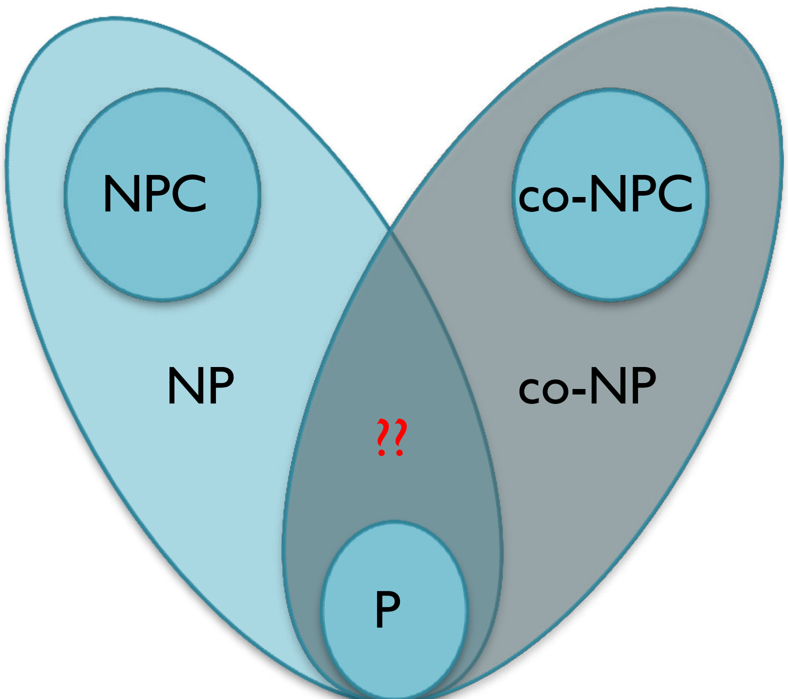
$$\Rightarrow \quad \begin{array}{l} NP \subseteq co\text{-}NP \\ NP = co\text{-}NP \end{array}$$

Let C_1 and C_2 be two complexity classes.

If $C_1 \subseteq C_2$, then $co\text{-}C_1 \subseteq co\text{-}C_2$.

Obs. $co\text{-}(co\text{-}C) = C$.

The diagram again



If an NP-complete language belongs to co-NP then

$$\text{NP} \subseteq \text{co-NP}$$
$$\Rightarrow \quad \text{NP} = \text{co-NP}$$

Let C_1 and C_2 be two complexity classes.

If $C_1 \subseteq C_2$, then $\text{co-}C_1 \subseteq \text{co-}C_2$.

Obs. $\text{co-(co-}C) = C$.

Integer factoring in NP ∩ co-NP

- Integer factoring.

  FACT = {(N, U): there's a prime in [U] dividing N}

- Claim. FACT ∈ NP ∩ co-NP

- So, FACT is NP-complete implies NP = co-NP.

Integer factoring in NP ∩ co-NP

- Integer factoring.

  FACT = {(N, U): there's a prime in [U] dividing N}

- Claim. FACT ∈ NP ∩ co-NP
- Proof. FACT ∈ NP : Give p as a certificate. The verifier checks if p is prime (AKS test), $1 \leq p \leq U$ and p divides N.

Integer factoring in NP ∩ co-NP

- Integer factoring.

  FACT = {(N, U): there's a prime in [U] dividing N}

- Claim. FACT ∈ NP ∩ co-NP
- Proof. FACT ∈ NP : Give the complete prime factorization of N as a certificate. The verifier checks the correctness of the factorization, and then checks if none of the prime factors is in [U].

Integer factoring in NP $\cap$ co-NP

- Integer factoring.

  FACT = {(N, U): there's a prime in [U] dividing N}

- Claim. FACT $\in$ NP $\cap$ co-NP
- Proof. FACT $\in$ NP : Give the complete prime factorization of N as a certificate. The verifier checks the correctness of the factorization, and then checks if none of the prime factors is in [U].

- Homework: If FACT $\in$ P, then there's a algorithm to find the prime factorization a given n-bit integers in poly(n) time.

Integer factoring in NP ∩ co-NP

- Integer factoring.

  FACT = {(N, U): there's a prime in [U] dividing N}

- Factoring algorithm. Dixon's randomized algorithm factors an n-bit number in $\exp(O(\sqrt{n} \log n))$ time.

Class EXP

- Definition. Class EXP is the exponential time analogue of class P.

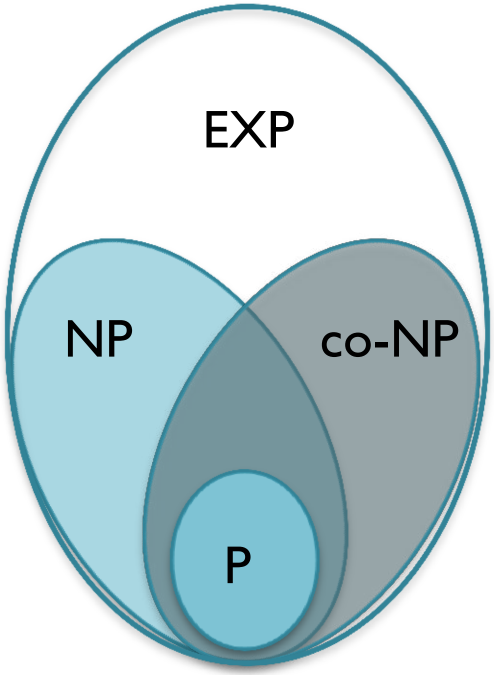
$$\mathrm{EXP} = \bigcup_{c \geq 1} \mathrm{DTIME}\left(2^{n^c}\right)$$

Class EXP

- **Definition.** Class EXP is the exponential time analogue of class P.

$$EXP = \bigcup_{c \geq 1} DTIME\left(2^{n^c}\right)$$

- **Observation.** $P \subseteq \boxed{NP \subseteq EXP}$



Class EXP

- Definition. Class EXP is the exponential time analogue of class P.

$$\mathrm{EXP} = \bigcup_{c \geq 1} \mathrm{DTIME}\left(2^{n^c}\right)$$

- Observation. P $\subseteq$ NP $\subseteq$ EXP

- *Exponential Time Hypothesis.* *(Impagliazzo & Paturi 1999)* Any algorithm for 3-SAT takes $\geq 2^{\delta . n}$ time, where $\delta > 0$ is some fixed constant and n is the no. of variables.

In other words, δ cannot be made arbitrarily close to 0.

Class EXP

- Definition. Class EXP is the exponential time analogue of class P.

$$\mathrm{EXP} = \bigcup_{c \geq 1} \mathrm{DTIME}\left(2^{n^c}\right)$$

- Observation. $P \subseteq NP \subseteq EXP$

- ***Exponential Time Hypothesis.*** *(Impagliazzo & Paturi 1999)* Any algorithm for 3-SAT takes $\geq 2^{\delta.n}$ time, where $\delta > 0$ is some fixed constant and n is the no. of variables.

ETH ➡ $P \neq NP$

Class EXP

- Definition. Class EXP is the exponential time analogue of class P.

$$\mathrm{EXP} = \bigcup_{c \geq 1} \mathrm{DTIME}\left(2^{n^c}\right)$$

- Observation. $\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{EXP}$

- *Exponential Time Hypothesis.* *(Impagliazzo & Paturi 1999)* Any algorithm for 3-SAT takes $\geq 2^{\delta . n}$ time, where $\delta > 0$ is some fixed constant and n is the no. of variables.

Homework: Read about Strong Exponential Time Hypothesis (SETH).