



# Computational Complexity Theory

## Lecture 5: NTMs; Class co-NP & EXP

Department of Computer Science,  
Indian Institute of Science

# Recap: Search version of NP

- 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.
- **Example:** Given a 3CNF  $\varphi$ , find a satisfying assignment for  $\varphi$  if such an assignment exists.

# 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.

# 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.

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

- **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” for deciding  $L_2$ .

# 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.

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

- **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” for deciding  $L_2$ .

Will be called an Oracle later

# 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.

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

- **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” for deciding  $L_2$ .

Karp reduction implies Cook reduction

# 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.

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

- **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” for deciding  $L_2$ .

Homework: Read about **Levin 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  $q_{\text{accept}}$  in addition to  $q_{\text{start}}$  and  $q_{\text{halt}}$ .

# 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  $q_{\text{accept}}$  in addition to  $q_{\text{start}}$  and  $q_{\text{halt}}$ .
- At every step of computation, the machine applies one of two functions  $\delta_0$  and  $\delta_1$  arbitrarily.



also called nondeterministically

# 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  $q_{\text{accept}}$  in addition to  $q_{\text{start}}$  and  $q_{\text{halt}}$ .
- At every step of computation, the machine applies one of two functions  $\delta_0$  and  $\delta_1$  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  $q_{\text{accept}}$  in addition to  $q_{\text{start}}$  and  $q_{\text{halt}}$ .
- At every step of computation, the machine applies one of two functions  $\delta_0$  and  $\delta_1$  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  $\delta_0$  and  $\delta_1$  (beginning from the start configuration) that makes  $M$  reach  $q_{\text{accept}}$ .
- **Definition.** An NTM  $M$  decides a language  $L \subseteq \{0,1\}^*$  if
  - $M$  accepts  $x \iff x \in L$
  - On every sequence of applications of the transition functions on input  $x$ ,  $M$  either reaches  $q_{\text{accept}}$  or  $q_{\text{halt}}$ .

# 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  $\delta_0$  and  $\delta_1$  (beginning from the start configuration) that makes  $M$  reach  $q_{\text{accept}}$ .
- **Definition.** An NTM  $M$  *decides* a language  $L \subseteq \{0,1\}^*$  if
  - $M$  accepts  $x \iff x \in L$
  - On every sequence of applications of the transition functions on input  $x$ ,  $M$  either reaches  $q_{\text{accept}}$  or  $q_{\text{halt}}$ .



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  $\delta_0$  and  $\delta_1$  (beginning from the start configuration) that makes  $M$  reach  $q_{\text{accept}}$ .
- **Defintion.** An NTM  $M$  *decides*  $L$  in  $T(|x|)$  time if
  - $M$  accepts  $x \iff x \in L$
  - On every sequence of applications of the transition functions on input  $x$ ,  $M$  either reaches  $q_{\text{accept}}$  or  $q_{\text{halt}}$  within  $T(|x|)$  steps of computation.



# Class NTIME

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

# Alternate characterization of NP

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

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

**Proof sketch:** Let  $L$  be a language in  $\text{NP}$ . 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  $\text{NTIME}(T(n))$  if there's an NTM  $M$  that decides  $L$  in  $c \cdot T(n)$  time on inputs of length  $n$ , where  $c$  is a constant.

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

**Proof sketch:** Let  $L$  be a language in  $\text{NP}$ . 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 guesses a  $u \in \{0, 1\}^{p(|x|)}$  by applying  $\delta_0$  and  $\delta_1$  nondeterministically

# Alternate characterization of NP

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

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

**Proof sketch:** Let  $L$  be a language in  $\text{NP}$ . 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  $\text{NTIME}(T(n))$  if there's an NTM  $M$  that decides  $L$  in  $c \cdot T(n)$  time on inputs of length  $n$ , where  $c$  is a constant.
- **Theorem.**  $\text{NP} = \bigcup_{c > 0} \text{NTIME}(n^c)$ .  
Proof sketch: Let  $L$  be in  $\text{NTIME}(n^c)$ . 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  $\text{NTIME}(T(n))$  if there's an NTM  $M$  that decides  $L$  in  $c \cdot T(n)$  time on inputs of length  $n$ , where  $c$  is a constant.

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

**Proof sketch:** Let  $L$  be in  $\text{NTIME}(n^c)$ . 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  $\text{NTIME}(T(n))$  if there's an NTM  $M$  that decides  $L$  in  $c \cdot T(n)$  time on inputs of length  $n$ , where  $c$  is a constant.

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

**Proof sketch:** Let  $L$  be in  $\text{NTIME}(n^c)$ . 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  $\delta_0$  and  $\delta_1$ .

**Class co-NP and EXP**



# Class co-NP

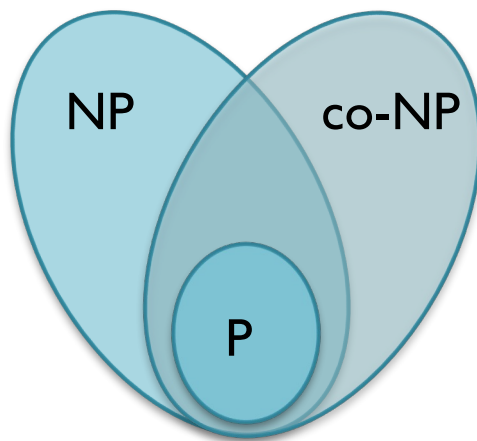
- **Definition.** For every  $L \subseteq \{0,1\}^*$  let  $\bar{L} = \{0,1\}^* \setminus L$ .  
A language  $L$  is in **co-NP** if  $\bar{L}$  is in **NP**.
- **Example.**  $\overline{\text{SAT}} = \{\varphi : \varphi \text{ is } \underline{\text{not}} \text{ satisfiable}\}.$

# Class co-NP

- **Definition.** For every  $L \subseteq \{0,1\}^*$  let  $\bar{L} = \{0,1\}^* \setminus L$ .  
A language  $L$  is in **co-NP** if  $\bar{L}$  is in **NP**.
- **Example.**  $\overline{\text{SAT}} = \{\varphi : \varphi \text{ is not satisfiable}\}.$
- **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  $\bar{L} = \{0,1\}^* \setminus L$ .  
A language  $L$  is in **co-NP** if  $\bar{L}$  is in **NP**.
- **Example.**  $\overline{\text{SAT}} = \{\varphi : \varphi \text{ is not satisfiable}\}.$



# Class co-NP

- **Definition.** For every  $L \subseteq \{0,1\}^*$  let  $\bar{L} = \{0,1\}^* \setminus L$ .  
A language  $L$  is in **co-NP** if  $\bar{L}$  is in **NP**.
- **Example.**  $\overline{\text{SAT}} = \{\varphi : \varphi \text{ is not satisfiable}\}.$
- **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 not 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

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

# 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

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

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

# 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

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

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

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

$\bar{M}$  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

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

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

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

  
 $\bar{M}$  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

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

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

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



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.**  $\overline{\text{SAT}}$  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  $L \in \text{co-NP}$ . Then  
 $\overline{L} \in \text{NP}$

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

**Proof.** Let  $L \in \text{co-NP}$ . Then

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

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

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

**Proof.** Let  $L \in \text{co-NP}$ . Then

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

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

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

# 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
$$\text{TAUTOLOGY} = \{\varphi : \text{every assignment satisfies } \varphi\}.$$
$$\text{TAUTOLOGY} \text{ is co-NP-complete.}$$

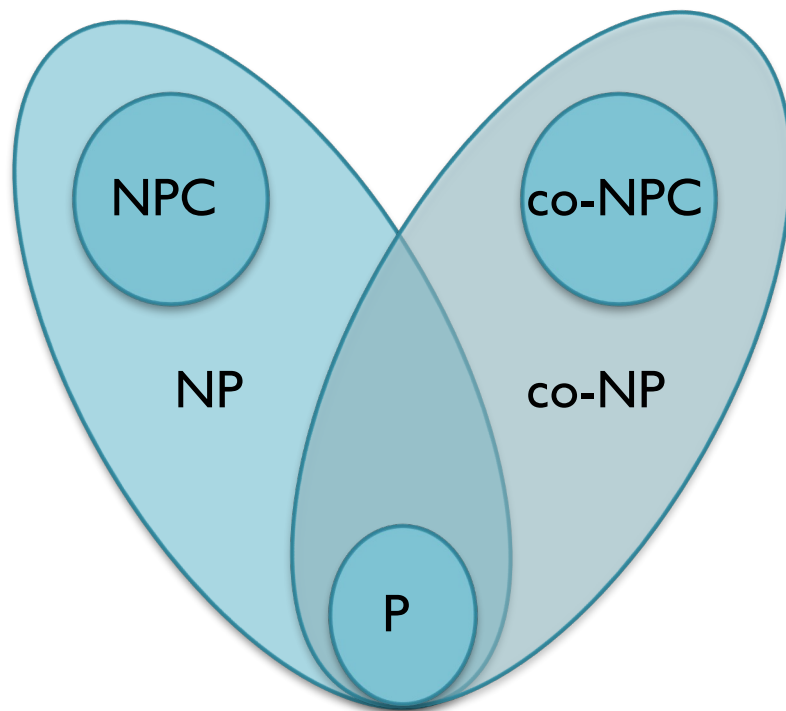
**Proof.** Similar (homework)

# 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  $\bar{L}$  is **co-NP-complete**  
**Proof.** Similar (homework)



# The diagram again



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

$$\begin{aligned} & \text{co-NP} \subseteq \text{NP} \\ \Rightarrow & \text{co-NP} = \text{NP} \end{aligned}$$

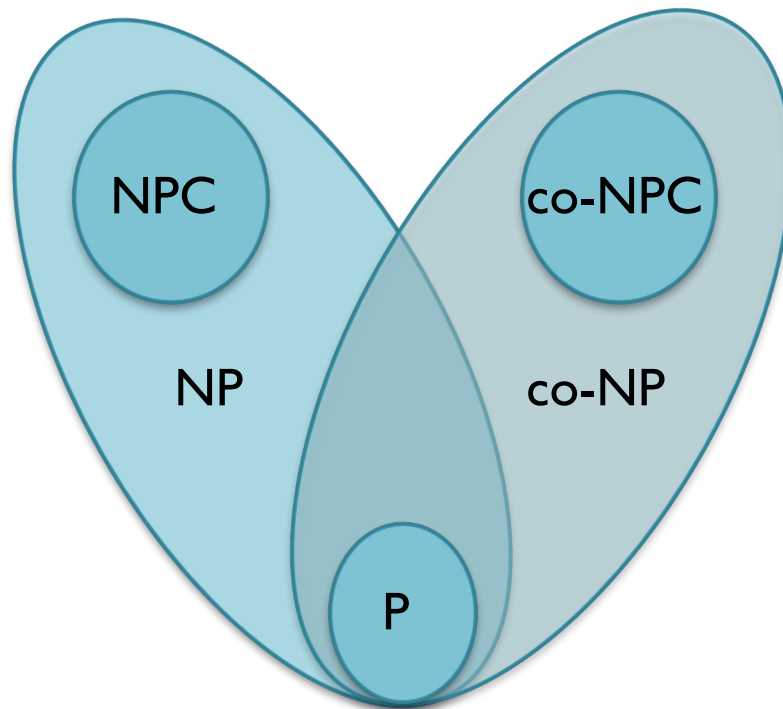


Let  $C_1$  and  $C_2$  be two complexity classes.

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

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

# The diagram again



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

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

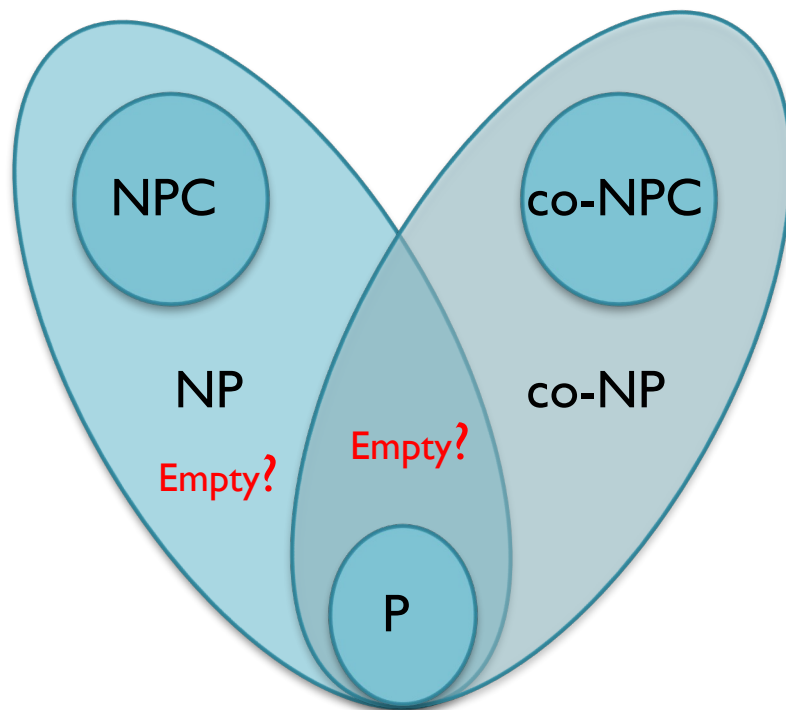


Let  $C_1$  and  $C_2$  be two complexity classes.

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

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

# The diagram again



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

$$\begin{aligned} \text{NP} &\subseteq \text{co-NP} \\ \text{NP} &= \text{co-NP} \end{aligned}$$

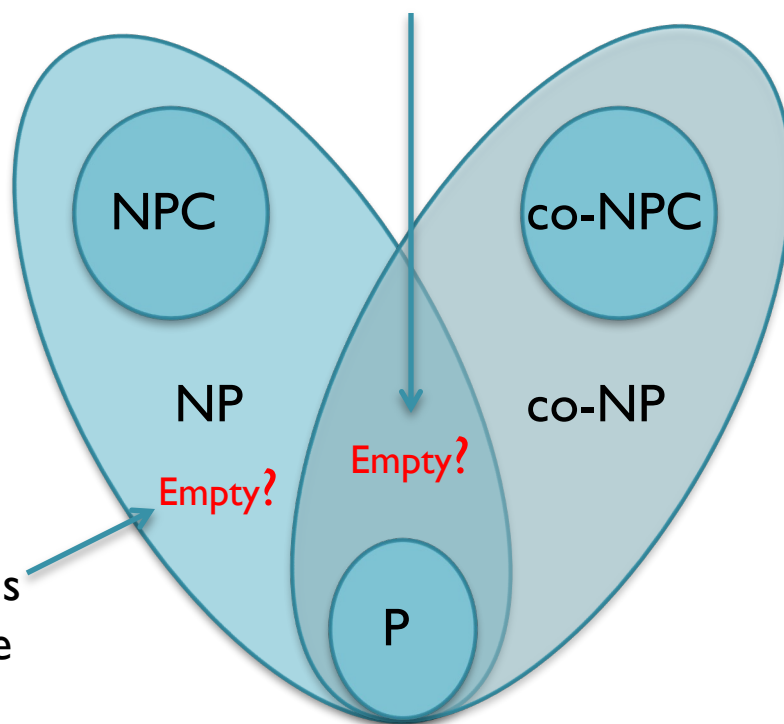
Let  $C_1$  and  $C_2$  be two complexity classes.

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

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

# The diagram again

We'll revisit this question.



We'll address this using a technique known as **diagonalization**

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

$$\begin{aligned} \text{NP} &\subseteq \text{co-NP} \\ \text{NP} &= \text{co-NP} \end{aligned}$$

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

# Integer factoring in $NP \cap co-NP$

- Integer factoring.

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

- Claim.  $FACT \in NP \cap co-NP$

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

# Integer factoring in $NP \cap co-NP$

- Integer factoring.

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

- Claim.  $FACT \in NP \cap co-NP$
- Proof.  $FACT \in NP$  : 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 $NP \cap co-NP$

- Integer factoring.

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

- 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  $[U]$ .

# Integer factoring in $NP \cap co-NP$

- Integer factoring.

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

- 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 $NP \cap co-NP$

- Integer factoring.

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

- 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.

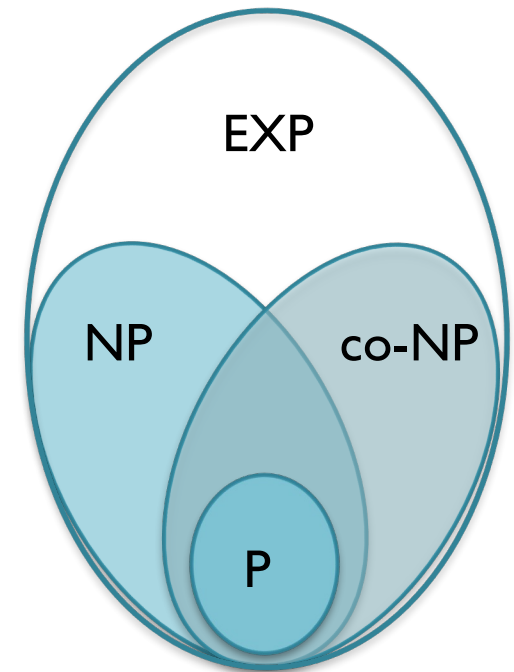
$$\text{EXP} = \bigcup_{c \geq 1} \text{DTIME} (2^{n^c})$$

# Class EXP

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

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

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



# Class EXP

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

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

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

 In other words,  $\delta$  cannot be made arbitrarily close to 0.

# Class EXP

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

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

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

$$\text{ETH} \Rightarrow P \neq NP$$

# Class EXP

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

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

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

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

# Class EXP

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

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

We'll address this using diagonalization

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

Is  $P \subsetneq \text{EXP}$  ?

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