Lecture 20:
coNP and Friends,
Oracles in Complexity Theory
Definition: $\text{coNP} = \{ L \mid \neg L \in \text{NP} \}$

**What does a coNP computation look like?**

In NP algorithms, we can use a “guess” instruction in pseudocode:

*Guess string $y$ of $|x|^k$ length...*

and the machine accepts iff some $y$ leads to an accept state.

In coNP algorithms, we can use a “try all” instruction:

*Try all strings $y$ of $|x|^k$ length...*

and the machine accepts iff every $y$ leads to an accept state.
Definition: A language B is coNP-complete if

1. $B \in \text{coNP}$

2. For every $A$ in coNP, there is a polynomial-time reduction from $A$ to $B$ (B is coNP-hard)
UNSAT = \{ \phi \mid \phi \text{ is a Boolean formula and no variable assignment satisfies } \phi \} 

Theorem: UNSAT is coNP-complete

TAUTOLOGY = \{ \phi \mid \phi \text{ is a Boolean formula and every variable assignment satisfies } \phi \} 
= \{ \phi \mid \neg \phi \in \text{UNSAT} \} 

Theorem: TAUTOLOGY is coNP-complete
Is $P = NP \cap coNP$?

THIS IS AN OPEN QUESTION!
An Interesting Problem in NP ∩ coNP

FACTORING
= \{ (m, n) \mid m > n > 1 \text{ are integers,}
    \text{there is a prime factor } p \text{ of } m \text{ where } n \leq p < m \}\}

If FACTORING ∈ P, we could expect to break most public-key cryptography currently in use!

Theorem: FACTORING ∈ NP ∩ coNP
FACTORING
= \{ (m, n) : m, n > 1 \text{ are integers, there is a prime factor } p \text{ of } m \text{ where } n \leq p < m \} \\

**Theorem:** FACTORING \( \in \) NP \( \cap \) coNP

**Proof:**

1. **FACTORING \( \in \) NP**
   - A prime factor \( p \) of \( m \) such that \( p \geq n \)
   - is a proof that \((m,n)\) is in FACTORING

2. **FACTORING \( \in \) coNP**
   - The prime factorization \( p_1^{e_1} \ldots p_k^{e_k} \) of \( m \) is a proof that \((m,n)\) is not in FACTORING:
     - Verify each \( p_i \) is prime, and \( p_1^{e_1} \ldots p_k^{e_k} = m \)
     - Verify that for all \( i=1,\ldots,k \) that \( p_i < n \)
Theorem: If $\text{FACTORING} \in P$, then there is a polynomial-time algorithm which, given an integer $n$, outputs either “$n$ is PRIME” or a prime factor of $n$.

Idea: Binary search for the prime factor!

Given binary integer $m$, initialize an interval $[2, m]$. If $(2, m)$ is not in FACTORING then output “PRIME” If $(\lceil m/2 \rceil, m)$ is in FACTORING then shrink interval to $[\lceil m/2 \rceil, m]$ else, shrink interval to $[2, \lceil m/2 \rceil]$

Keep picking $n$ to halve the interval with each call to FACTORING. Takes $O(\log n)$ calls to FACTORING!
NP-complete problems:

SAT, 3SAT, CLIQUE, VC, SUBSET-SUM, ...

coNP-complete problems:

UNSAT, TAUTOLOGY, NOHAMPATH, ...

(NP \cap coNP)-complete problems:

Nobody knows if they exist!

P, NP, coNP can be defined in terms of specific machine models, and for every possible machine we can give an encoding of it.

NP \cap coNP is not known to have a corresponding machine model!
Polynomial Time
With Oracles

*We do not condone smoking. Don’t do it. It’s bad. Kthxbye
Oracle Turing Machines

Is formula $F$ in SAT?

$q_{YES}$

yes

INPUT

INFINITE TAPE
Oracle Turing Machines

An oracle Turing machine $M^B$ is equipped with a set $B \subseteq \Gamma^*$ to which a TM $M$ may ask membership queries on a special “oracle tape”

[Formally, $M^B$ enters a special state $q_?$]

and the TM receives a query answer in one step

[Formally, the transition function on $q_?$ is defined in terms of the entire oracle tape:
if the string $y$ written on the oracle tape is in $B$, then state $q_?$ is changed to $q_{\text{YES}}$, otherwise $q_{\text{NO}}$]

This notion makes sense even when $M$ runs in polynomial time and $B$ is not in $P$!
How to Think about Oracles?

Think in terms of Turing Machine pseudocode!

An oracle Turing machine $M$ with oracle $B \subseteq \Gamma^*$ lets you include the following kind of branching instructions:

“if ($z$ in $B$) then <do something> else <do something else>”

where $z$ is some string defined earlier in pseudocode. By definition, the oracle TM can always check the condition ($z$ in $B$) in one step.
Some Complexity Classes With Oracles

Let $B$ be a language.

$P^B = \{ L \mid L \text{ can be decided by some polynomial-time TM with an oracle for } B \}$

$P^{\text{SAT}} = \text{the class of languages decidable in polynomial time with an oracle for SAT}$

$P^{\text{NP}} = \text{the class of languages decidable by some polynomial-time oracle TM with an oracle for some } B \text{ in NP}$
Is $P^{SAT} \subseteq P^{NP}$?

Yes! By definition...

Is $P^{NP} \subseteq P^{SAT}$?

Yes!

Every NP language can be reduced to SAT!

Let $M^B$ be a poly-time TM with oracle $B \in NP$. We define $N^{SAT}$ that simulates $M^B$ step for step. When the sim of $M^B$ makes query $w$ to oracle $B$, $N^{SAT}$ reduces $w$ to a formula $\phi_w$ in poly-time, then calls its oracle for SAT on $\phi_w$. 
Is $NP \subseteq P^{NP}$?

Yes!

*Just ask the oracle for the answer!*

For every $L \in NP$ define an oracle TM $M^L$ which asks the oracle if the input is in $L$, and outputs the answer.
Is coNP ⊆ P^{NP}?

Yes!

Again, just ask the oracle for the answer!

For every \( L \in \text{coNP} \) we know \( \neg L \in \text{NP} \)

Define an oracle TM \( M^{\neg L} \) which asks the oracle if the input is in \( \neg L \)

- \textit{accept} if the answer is no,
- \textit{reject} if the answer is yes

In general, \( P^{NP} = P^{\text{coNP}} \)
For every poly-time TM M with oracle $B \in P$, we can simulate every query $z$ to oracle $B$ by simply running a polynomial-time decider for $B$.

Is $P^B \subseteq P$?

Yes!

For every poly-time TM M with oracle $B \in P$, we can simulate every query $z$ to oracle $B$ by simply running a polynomial-time decider for $B$.

The resulting machine runs in polynomial time!
\[ \mathsf{P}^\mathsf{NP} = \text{the class of languages decidable by some polynomial-time oracle TM } M^B \text{ for some } B \text{ in NP} \]

**Informally:** \( \mathsf{P}^\mathsf{NP} \) is the class of problems you can solve in polynomial time, assuming SAT solvers work.
$P^{NP} =$ the class of languages decidable by some polynomial-time oracle TM $M^B$ for some $B$ in NP

Informally, $P^{NP}$ is the class of problems you can solve in polynomial time, if SAT solvers work.

A problem in $P^{NP}$ that looks harder than SAT or TAUT:

$\text{FIRST-SAT} = \{ (\phi, i) \mid \phi \in \text{SAT} \text{ and the } i\text{th bit of the lexicographically first SAT assignment of } \phi \text{ is } 1 \}$

Using polynomially many calls to SAT, we can compute the lex. first satisfying assignment.
Is $\text{NP}^\text{B} = \text{NP}$?

Is $\text{coNP}^\text{B} = \text{coNP}$?

$\text{NP}^\text{B} = \{ L \mid L \text{ can be decided by a polynomial-time}\,$
\begin{center}
\text{nondeterministic} \quad \text{TM with an oracle for B} \}\$

$\text{coNP}^\text{B} = \{ L \mid L \text{ can be decided by a poly-time}\,$
\begin{center}
\text{co-nondeterministic} \quad \text{TM with an oracle for B} \}\$

Is $\text{NP} = \text{NP}^{\text{NP}}$?

Is $\text{coNP}^{\text{NP}} = \text{NP}^{\text{NP}}$?

\textbf{THESE ARE OPEN QUESTIONS!}

It is believed the answers are NO ...
Logic Minimization is in $\text{coNP}^{\text{NP}}$

Two Boolean formulas $\phi$ and $\psi$ over the variables $x_1, \ldots, x_n$ are equivalent if they have the same value on every assignment to the variables.

Are $x$ and $x \lor x$ equivalent? Yes

Are $x$ and $x \lor \neg x$ equivalent? No

Are $(x \lor \neg y) \land \neg (\neg x \land y)$ and $x \lor \neg y$ equivalent? Yes

A Boolean formula $\phi$ is minimal if no smaller formula is equivalent to $\phi$

MIN-FORMULA = \{ $\phi$ | $\phi$ is minimal \}
Theorem: \( \text{MIN-FORMULA} \in \text{coNP}^\text{NP} \)

Proof:

Define \( \text{NEQUIV} = \{ (\phi, \psi) \mid \phi \text{ and } \psi \text{ are not equivalent} \} \)

Observation: \( \text{NEQUIV} \in \text{NP} \) (Why?)

Here is a \( \text{coNP}^{\text{NEQUIV}} \) machine for \( \text{MIN-FORMULA} \):

Given a formula \( \phi \),

Try all formulas \( \psi \) such that \( \psi \) is smaller than \( \phi \).

If \( ((\phi, \psi) \in \text{NEQUIV}) \) then accept else reject

Note: \( \text{MIN-FORMULA} \) is not known to be in \( \text{coNP} \)!
The Difficulty of Formula Minimization

MIN-CNF-FORMULA = \{ \phi \mid \phi \text{ is CNF and is minimal} \}

**Theorem**: MIN-CNF-FORMULA is $\text{coNP}^{\text{NP}}$-complete

**Proof**: Beyond the scope of this course...

**Note**: We don’t know if MIN-FORMULA is $\text{coNP}^{\text{NP}}$ complete!
FACTORING

NP

coNP

P

SAT

FACTORIZATION

TAUT

MIN-FORMULA

P

NP

coNP

NP

NP

coNP

NP

NP
Oracles and P vs NP

Theorem:

(1) There is an oracle B where $P^B = NP^B$

(2) There is an oracle A where $P^A \neq NP^A$

See Sipser 9.2

Moral: Any proof technique that extends to Turing Machines with arbitrary oracles won’t be able to resolve P versus NP!